University of Cagliari

PhD School of Mathematics
and Computer Science

XXVII cycle

Course in Computer Science

Structured Meshes:

Composition and Remeshing guided by the Curve-Skeleton

INF/01

Author:
Francesco Usai

Supervisor:
Prof. Riccardo Scateni

PhD Coordinator:
Prof. G. Michele Pinna

2014 - 2015

Acknowledgements

Francesco Usai gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.)

Abstract

Virtual sculpting is currently a broadly used modeling metaphor with rising popularity especially in the entertainment industry. While this approach unleashes the artists' inspiration and creativity and leads to wonderfully detailed and artistic 3D models, it has the side effect, purely technical, of producing highly irregular meshes that are not optimal for subsequent processing. Converting an unstructured mesh into a more regular and structured model in an automatic way is a challenging task and still open problem.

Since structured meshes are useful in different applications, it is of interest to be able to guarantee such property also in scenarios of part based modeling, which aim to build digital objects by composition, instead of modeling them from a scratch.

This thesis will present methods for obtaining structured meshes in two different ways. First is presented a coarse quad layout computation method which starts from a triangle mesh and the curve-skeleton of the shape. The second approach allows to build complex shapes by procedural composition of PAM's. Since both quad layouts and PAMs exploit their global structure, similarities between the two will be discussed, especially how their structure has correspondences to the curve-skeleton describing the topology of the shape being represented. Since both the presented methods rely on the information provided by the skeleton, the difficulties of using automatically extracted curve-skeletons without processing are discussed, and an interactive tool for user-assisted processing is presented.

Contents

Introduction

The predominant way to represent digital objects is an approximation of their exterior surface using polygonal cells connected together, forming a polygonal mesh. The preferred polygon for surface representations has always been the triangle, making the triangle mesh representation ubiquitous in every application which makes use of computer graphics techniques. Due to their simplicity, however, triangle meshes do not encode any global structure of the object being represented.

Over the years the approaches for the construction of digital objects have changed. One of the main approaches for digital object modeling has always been the *Polygonal Modeling* in which the 3D artist directly manipulates the polygonal structure that is going to compose the shape she has in mind. Common workflows used within this approach allows the artists to design their objects following their ideal global structure starting from hexahedral boxes that are iteratively extruded and refined. Nowadays, especially for character modeling, the virtual sculpting metaphor gained a widespread diffusion and it allows 3D artists to free their creativity with a more natural modeling metaphor. Modeling a digital object or an animation character through an approach similar to clay sculpting is more intuitive than directly manipulating the elements of its surface. Due to the high density and highly irregular triangle meshes built using this approaches, the task of obtaining structured models from the unstructured ones is still complex and requires a significant user intervention. The same issues arise with 3D models that have been obtained by acquiring a real world object using 3D scanners.

Great part of the efforts of the research in the field of shape modeling are directed towards providing the practitioners with tools that can unleash their creativity and free them from the tedious and complex parts of the job.

Different approaches have been introduced such the conversion of free-hand drawings into 3D meshes, the virtual sculpting metaphor, part based modeling and procedural modeling; all with the purpose of simplify the modelers' work and allow them to obtain better results with less efforts.

However, the task of converting an irregular mesh usually produced by the aforementioned methods, to a structured one, a task known with the

term *Retopology* or *Remeshing*, it is known to be a time consuming and challenging task, even for experienced professionals due to the complexity of managing an object's global structure with the use of local operations. While there have been major improvements in this topic, great part of the work is still done by hand, thus requiring a great deal of time and efforts to produce high quality results. This complexity harms, to some extent, the benefits of having simpler ways to obtain high quality models, shifting the efforts from the task of creating the initial high resolution model to the task of defining its global structure.

This issue have been addressed in different ways and a number of approaches have been proposed; a great part of them work in a full automatic manner [BCE*13] processing geometrical information in order to reveal a higher level structure of the mesh. However, in spite of the many attempts to automatize this process, there still exists no satisfactory methodfor practical applications, since the results do not capture the high-level features required for editing, deformation, or animation.

The main idea of this thesis is that structured meshes can be obtained with the guidance of a descriptor able to encode the higher level information needed to recover this structure, that geometry alone is not able to convey. The shape descriptor of choice is the curve-skeleton, which is able to encode topological, geometrical and volumetric information, that are key aspects when trying to recover the global structure of a given object.

In this thesis will be presented two approaches to address this issue from two different angles: the first one tries to reveal the global structure of a given digital object represented as a unstructured mesh, while the other proposes a way to obtain structured meshes by construction.

The first approach, presented in Part III allows to extract the global structure of a given object represented as triangular mesh computing its coarse quad layout using the information provided by the curve-skeleton as a guidance. The presented method is fully automatic, but supports user interaction techniques to refine the automatically obtained result, with the use of an intermediate structure that allows an intuitive manipulation of the global structure.

In Part IV an approach for mesh composition based on the Polar-Annular Mesh (PAMs) representation [BAS14] is presented. The PAM representation encodes the skeleton of the shape, and the approach being proposed allows to procedurally compose new shapes, starting from a user defined set of parts, guaranteeing that the resulting object is still represented as a structured mesh which encode its global structure. During the composition process, the encoded skeleton is used both for determining how to connect the different parts, and to avoid self intersections.

Moreover, since the curve-skeleton is a key ingredient of the two presented approaches, in Part II the issues of automatically extracted skeletons are discussed and a newly developed tool for their interactive processing is

presented. Its application appears to be well suited for cleaning and repairing the results of the skeletonization approaches in order to make them fit given application pipelines allowing to obtain optimal results.

Part I

Background

Chapter 1

Curve-Skeletons

Nowadays the use of digital representation of real world objects an figments of fantasy or artistic processes is abundant and pervasive. The use of these representations, if previously confined to a few niches, is currently present in a variety of applicative fields ranging from the entertainment industry to medical applications, from architectural design to immersive virtual reality environments, from cultural heritage to industrial production processes.

Having this incredibly high availability of 3D models, there is also a high demand for compact and expressive ways to represent them. These representations are known as **shape descriptors**. Compact shape descriptors allow for efficient storage, comparison and processing, since they should encode only the salient information and features of the a shape. A tradeoff between compactness and level of detail is usually required. At one hand the higher the compactness, the lower will be the number of features that can be encoded. On the other hand, an higher level of detail requires more storage and processing time, but should allow for a more precise description of the salient features of a shape.

Skeletons are one of the most widely adopted shape descriptors, in fact they are used in a wide range of applications such has, to name a few: computer animation [BP07] , shape matching [HSKK01], modelling [BAS14], remeshing and quad layout extraction [Usa15], polycube [LZLW15] and hexahedral mesh construction [ZBG*07a]. They provide a compact representation for meaningful information about both volume and topology of a shape. The following sections will discuss 1.1 the definition of curve-skeletons both in the 2D and in the 3D case, 1.2 the major techniques for extracting a curve skeleton from a 3D shape, while 1.4 will discuss some of the applications in which they are used and 1.5 a list of the desired features.

1.1 Definitions

The concept of skeleton directly derives from the concept of *Medial Axis Transform* that was firstly introduced by H.Blum in 1967 [Blu67] as a convenient shape descriptor able to represent the features of a planar shape.

The Medial Axis Transform (MAT) for a planar object $\mathcal{O}$ with boundary $\partial\mathcal{O}$ is defined as the locus of centres of its maximal inscribed disks. A disk $D_i \subset \mathcal{O}$ is denoted by its center p and its radius r and is called maximal if, for any other disk D_j, if D_i is contained in D_j then D_i and D_j are the same disk. Each maximal disk is tangent to $\partial\mathcal{O}$ in two or more points equidistant to the disk center, exploiting the fact that the skeleton of a shape can be seen also as a composition of its symmetry axes.

As previously pointed out, each maximal disk is described not only by its center, but also by its radius that is what makes the medial axis a transform, since these two information about each point of the skeleton allow to invert the process and reconstruct the original shape. If we consider only the position of each point, we will retain only the graph representation, able to encode topology and medial axis of the considered shape.

A good way to intuitively understand how the medial axis of a shape can be determined and how it reveals the symmetries of a shape is the **Grassfire analogy**. If we consider our object $\mathcal{O}$ as a patch of grass, and we imagine to set its boundary $\partial\mathcal{O}$ on fire, this fire will propagate isotropically towards the interior of $\mathcal{O}$ with uniform speed along the anti-normals of the boundary. At some point in time, some of the fire fronts coming from different directions will meet and extinguish. When all the fire fronts have quenched, all the lines where they have met will form the medial axis of our shape $\mathcal{O}$.

3D Skeletons

The definition of the medial axis transform for the 2D setting, can be directly extended to the 3D case requiring only to define it with respect to the maximal inscribed spheres instead of the maximal inscribed disks. Unfortunately extending Blum's definition to the n-dimensional case, the medial representation will be composed of different parts whose dimensionality is up to $n-1$. According to this property, calculating the medial axis of a 3D shape will result in a collection of connected curves and sheets, which are impractical to use in real world applications.

A mono-dimensional structure results to be a much more simpler and intuitive representation, that is also easier and natural to manipulate. However a precise and universally accepted definition of *Curve-Skeleton* is still lacking. Only Dey and Sun have tried to bridge this gap with the definition of the *medial geodesic skeleton* [DS06], which can be intuitively defined as the medial path of the medial axis of a 3D shape.

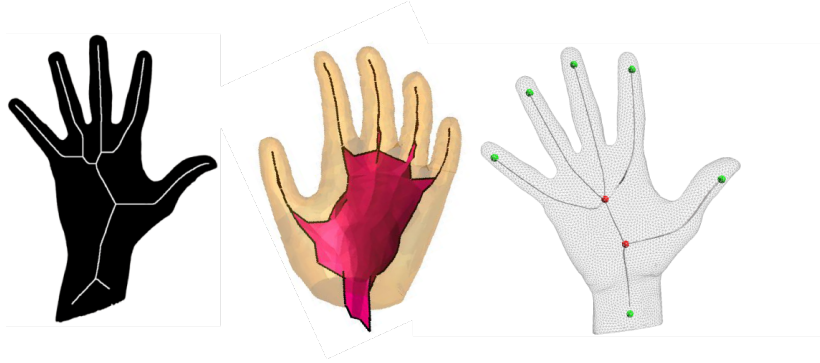


Figure 1.1: *The skeleton (or meadial axis) of a 2D shape (left) compared with its extension in the 3D case (center) and the mono-dimensional skeletal structure of a curve-skeleton (right).*

In order to give a more formal definition of this concept we need to consider our 3D shape $\mathcal{O}$ whose boundary $\partial\mathcal{O}$ is a *connected manifold* surface S. Following the 3D extension of the Blum's definition we define the medial axis M as the set of the centers of the maximal spheres inscribed in $\mathcal{O}$. In a previous work [GK04], Giblin and Kimia , showed that M can contain only five different types of points. One type forms two dimensional sheets (the 2D part of a medial axis), two of the types form the curves and the other two types are isolated points on the medial axis. Figure 1.2 shows four of these five types. Dey and Sun [DS06] denoted the set of all the points belonging to the first type with M^2. Each point $x \in M_2$ (the grey sheets in 1.2) have the property that its associated maximal sphere B_x is bitangent to S at points a_x and b_x. They defined a function $f(x)$ to be the length of the shortest medial path between a_x and b_x, and called it *Medial Geodesic Function* (MGF). The MGF has the property of being continuous and differentiable everywhere on M_2 except at some singular points that in practice occur *roughly medially* with respect to M_2 and which form the curve-skeleton in M_2, called Sk_2.

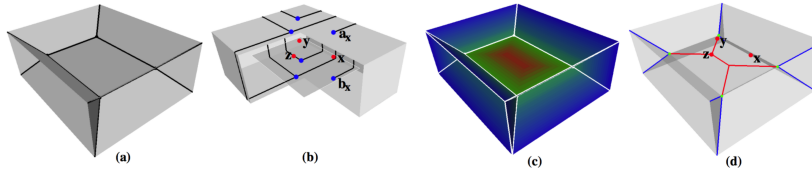


Figure 1.2: *(a) medial axis of a rectangular block. (b) one sheet of the medial axis (c) the medial axis and the color coded MGF values. (d) red and blue lines and green points are respectively Sk_2, Sk_3 and the limit points of their union.*

In order to define the curve-skeleton for the entire object $\mathcal{O}$ it is necessary to consider also the set of points $M_3 \subset M$ belonging to the curves generated by the intersection of 3 sheets in M_2. The points belonging to these curves have the property that their maximal spheres touch the surface S at exactly three points. The skeleton Sk_3 in M_3 is composed of all the points x for which the gradient of the MGF for the three meeting sheets sinks into x. The *Medial Geodesic Skeleton* of $\mathcal{O}$ is then the closure of $Sk_2 \cup Sk_3$.

1.2 Extraction techniques

During the last two decades, several automatic skeletonization methods have been presented. A common way to categorize these skeleton extraction methods is based on their input that can be either a voxel-based representation, a mesh or a point cloud. The following sections will briefly present the state of the art methods, accordingly to this categorization.

Voxel-based methods

Volumetric representations of shapes rely on the concept of *Voxel* (volume element), that is a point $p(x, y, z)$ in the discrete space $\mathbb{R}^3$ where $x, y, z \in \mathbb{Z}$. Each voxel can be seen as a cube of unitary volume, which is composed of 6 faces, 12 edges and 8 corners. As for the pixels of an image, we can define the concept of neighbourhood of a voxel in different ways. When we talk about pictures, we say that two pixels are 4-neighbours if they share an edge or 8-neighbours if they share an edge or a corner. In the same way we can define the neighbour relationship for voxels denoting as 6-neighbours two voxels that share a face, 18-neighbours if they share a face or an edge, 26-neighbours if they share a face, an edge or a corner. We can then define the n-neighbourhood of a voxel as the set of its n-neighbours, where n can be either 6, 18 or 26.

Skeleton extraction algorithms working on volumetric representation try to produce curve-skeletons iteratively applying a *thinning* operator, whose effect is to remove voxels from the boundary of the object until there are no more *simple points* to remove. According to [Mor81] a simple point is defined as a voxel that can be removed from an object keeping its topology unchanged. It is important to note that simple point detection strictly depends on the choice of using a 6, 18 or 26-neighbourhood and it is possible to determine if a voxel is a simple point or not, just inspecting its neighbourhood.

A number of algorithms belonging to this category, such as [MS96, WB07], rely on predefined templates, if the neighbourhood of a voxel matches one of the templates it can be removed. In [ZLLJ08] thinning is applied on the parts produced by the decomposition of the initial shape and the skeletal

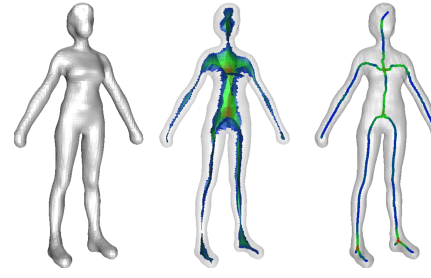
branches are then stitched together. Other approaches rely on defining a discretization of a continuous function which construct a scalar or vector fields in the discrete space, building the skeleton upon their singularities. The thinning process can be also guided by some measure of the significance of its composing voxels, as in [ABZ*] where a measure called the *medial persistence* is defined.

The most used function for the field generation is the *distance transform*, which relies on approximated definitions of the euclidean distance. In [Bor96] has been proved that the best discrete approximation is the (3, 4, 5) distance where the distance between two adjacent voxels can only assume the values 3, 4 or 5 respectively when the two voxels share a face, an edge or a vertex. The distance transform is not the only possible choice, in fact Cornea and colleagues [CSYB05] used the Newtonian potential model, using a force vector field inside the object by applying a charge on its boundary.

Mesh-based methods

Meshes have always been the most used way to represent a 3D object, hence it follows that a vast and heterogeneous majority of the skeletonization algorithms are mesh-based.

An important class of these approaches based the calculation of the skeleton of a shape processing its medial axis. As presented in the previous section the *Medial Geodesic Skeleton* defined in [DS06], whose pipeline is depicted in figure **??**, falls into this category.



Despite the fact that the work of Dey and Sun has a big theoretical relevance, since a precise and sound definition of curve-skeleton was given for the first time, the skeletonization algorithm derived from their definition has some defects. One of the main issues is the long computational time, since it requires the computation of the geodesic paths between several point pairs. Moreover there are stability issues, due to the well known sensitivity of the medial axis to noise and small perturbations of the object's surface. This sensitivity leads to noisy skeletal paths and spurious branches.

The major requirement for a curve-skeleton is to be a mono-dimensional structure, hence a solid with *zero* volume. While voxel based methods achieve this result using the thinning operator, mesh based methods must use a different approach to iteratively shrink the volume bounded by the mesh surface. In [ATC*08] the iterative volume reduction of a shape is per-

formed by iteratively contracting the shape's surface, moving its geometry inward accordingly to its *mean curvature flow* (MCF). The process can be seen on Figure 1.3. In their approach, and in subsequent variations of it, the

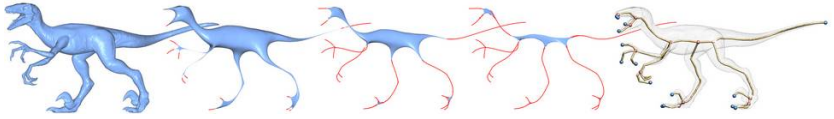
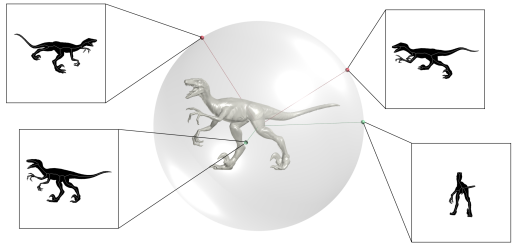


Figure 1.3: *Process of mesh contraction. The structure exploited in the intermediate steps is called* meso-skeleton. *Image courtesy of [ATC*08].*

MCF computation needs to be constrained in order to preserve the topological information, since a unconstrained contraction would make the shape collapse into a single point and not into a nice mono-dimensional skeletal structure. This method was a significant advance since the constrained MCF can be encoded as a system of linear equations that can be efficiently solved and it showed to be a robust framework, less sensible to noise or low quality of the triangulations. An important extension to this approach is presented in [TAOZ12] which simplifies the problem definition, also from the end-user point of view, and allow to obtain the so-called *meso-skeletons* which are intermediate structures between medial axis and curve-skeletons.

Notable exceptions to this trend are [LGS12, LS13, KJT13] which compute the curve-skeleton of a 3D shape as the composition of a set of 2D MAT. The pipeline of these methods can be summarized as follows. A set of 2D silhouettes of the shape are generated and for each of them the 2D skeleton is computed. Each point of each 2D skeleton is then projected back to the 3D space voting a voxel grid. Each point is then projected back to all the other silhouettes in order to check if it is an artifact due to occlusion or not; if it is, it will not be part of the resulting grid. The remaining points are then used to build a graph that is then processed in order to build the final structure. While these approaches are less robust than [TAOZ12] the main advantage is that, relying on the external appearance of an object rather than the structure of its surface representation, they are resolution independent. Moreover these methods are also able to produce curve skeletons from non manifold meshes and point clouds. Another important aspect is that they present heuristics for collapsing spurious branches and closing loops whenever two terminal nodes have intersecting maximal balls.



Point-based methods

Sometimes a well organized volumetric or surface representation is not available, that is the case of the data produced by the devices used to acquire real objects' geometry, such as range scanners or other low cost devices such as Microsoft Kinect. Those dataset can be composed of points that can encode or not the orientation of the surface they represent. Some of these approaches took inspiration from successful methods of the other categories. In [CTO*10] a contraction method is defined using a different formulation that is able to work on point data without connectivity, instead of triangle meshes.

The method presented in [TZCO09] assume that the input shape is mostly composed of tube-like parts which are detected and processed separately, producing their rotational symmetry axis (ROSA) which the authors observed to be the medial path of cylindrical parts. These symmetry axis are then merged together in order to obtain the resulting curve-skeleton.

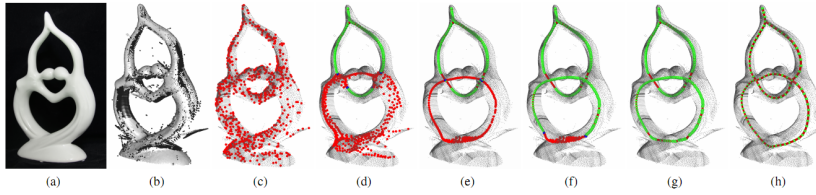


Figure 1.4: *Pipeline of the L1 approach proposed in [HJS* 14].*

In [HJS*14] Huang and colleagues presented the L1-Medial Skeleton method which operates on raw scan data composed of non-oriented points (Figure 1.4a). The methods firstly downsample the original point data (Figure 1.4b), and each sampled point is then iteratively reprojected and redistributed with respect to the center of the input and its local neighbourhood. When the points form curve like structures, they are assembled into skeletal branches (Figure 1.4c-h). The size of the used neighbourhood is increased at each step in order to capture features at different levels of detail. This method have shown to be more robust and accurate of the previous ones, producing high quality curve-skeletons even when data is incomplete or significantly affected to noise.

1.3 Inverse Skeletonization

As the term *Skeletonization* refers to the process of computing a curve-skeleton from a 3D object, whatever its representation, we can define the process of building a 3D shape from which the input skeleton could have been extracted as *Inverse Skeletonization*.

In recent years, some inverse skeletonization methods have been presented as well as commercial solutions, usually integrated into complete 3D modelling softwares. All these approaches take as input a curve-skeleton with, associated to each skeletal node, the radius of the inscribed maximal sphere, in order to express the local volume of the shape. These approaches are partially inspired by a previous work of Srinivasan et al. [SMA] which presented a novel method for creating an assembly of pipes from a wire-frame mesh. Although the paper do not refer to skeletons, it can be seen as a common ancestor of the current approaches, even if it is thought for architectural or blocky structures, while the current approaches are mainly focussed on organic shapes or animal-like creatures.

The major differentiation between inverse skeletonization approaches is based on the strategy used to build the output mesh. Practically speaking, building the mesh around tubular parts is easy, while building it around the branching nodes is a more complex task.

One of the strategies is to start with the tubes and then build the mesh at the branching nodes of the skeleton, while the other works in the opposite way, building at first the mesh around the branching nodes, then build the tubes.

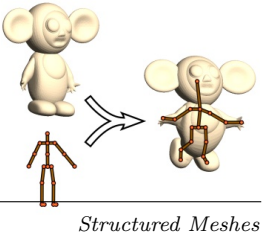
B-Mesh [JLW10] belongs to the first category, in fact the approach relies on building the tubular mesh around the skeletal branches and then stitch them at the branching node. Belonging to the other category is the approach called SQM presented in [BMW12] where the mesh around the branching nodes are first modelled, and the tubular parts are then extruded. Both the approaches produce meshes that contain both triangles and quadrilaterals, however there are some differences since B-Mesh aims to produce meshes composed of quadrilaterals only, but it is not always able to obtain such result, while SQM produces by construction mesh composed of only quadrilaterals on the tubular parts and triangle fans at the polar caps.

1.4 Applications

The ability of curve-skeletons of concisely encode both topological and structural information of a three-dimensional object is beneficial to many applicative fields; this section will briefly present some of the them.

Animation

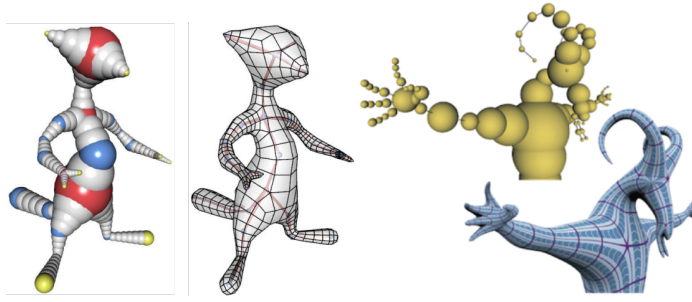
The essential task of computer animation is to process a digital character in order to make it assume the desired pose. Directly manipulating the object's geometry can be time-consuming, tedious and error prone, hence different methods for posing



digital characters, manipulating a simpler proxy structure, were developed over time. Two main methods have emerged as the most commonly used : *Control Cages* and *Skeletal Subspace Deformation*. While the first one uses a simple mesh (the *Cage*) surrounding the object, the latter uses a skeletal representation (*Kinematic Skeletons*) to move the object's geometry accordingly to the transformations applied to its points. Kinematic skeletons have shown to be a natural way to define the pose of the character's body, since they can mimic the different bones of an anthropomorphic or a zoomorphic creature. They can be obtained subsampling a curve-skeleton keeping only the points in locations where the user wants the articulation of the body to happen.

Shape Matching

The process of *Shape Matching* aims to determine if two different objects, roughly or precisely, are the same and give a measure of their sameness. Trying to find correspondences between two shapes operating directly on their 3D representation can be quite hard. This task can benefit from the use of shape descriptors able to encode concisely meaningful information about the shapes being compared. A curve-skeleton representation is a good candidate for shape matching due its ability of representing the shape as a graph of its logical components and its homotopy with the shape. Graph based approaches to shape matching have been quite successful since they reduce the dimensionality of the problem, since the number of components of a skeletal graph structure is much smaller than the number of geometric components of a polygonal mesh, polyhedral mesh or volumetric representation.



(a) *[JLW10]* **(b)** *[BMW12]*

Figure 1.5: *Two examples of inverse skeletonization approaches.*

Modelling

In recent years, with the popularity rise of digital sculpting tools, there have been a substantial interest into being able to quickly generate base shapes that will be further defined with these tools. A widely used approach is to allow the user to build a skeleton of the shape she wants to create and build the surface mesh that wraps it, using an inverse skeletonization approach. This possibility has recently been added to commercial modelling softwares like *ZBrush*, which presents is tool called ZSpheres [ZBr15] and Autodesk with its tool *123d Creatures* [Aut15].

1.5 Desired Properties

This section will briefly present a set of desirable properties of a curve-skeleton, which were discussed for the first time in [CSM07]. While considering these desired properties, it is important to be aware that some of the features can be in contrast with each other, and each applicative context can have a specific subset of features that are important or essential.

- **Homotopy**:
  The curve-skeleton of an object should be topologically equivalent to the object itself, meaning that they should have the same number of connected components and handles.

- **Invariance under isometric transformations**:
  Let T be an isometric transformation, $\mathcal{O}$ a 3D object and $Sk(\mathcal{O})$ its curve-skeleton, $T(Sk(\mathcal{O})) = Sk(T(\mathcal{O}))$ must hold. In simpler terms can be stated that the curve-skeleton of an object transformed using the transformation T should be equal to the transformed curve-skeleton of the original object. This is an important property for shape matching applications since it can be necessary to compare curve-skeletons of objects with different size and orientations.

- **Reconstruction**:
  In the 2D setting the MAT is able to reconstruct the original object as the union of all the maximal disks. This is a desirable property also for curve-skeletons of 3D objects, however being the curve-skeleton only a subset of the MAT of a 3D object, the union of all its maximal spheres does not guarantee a complete reconstruction. Possible approaches to reconstruction can be using maximal inscribed ellipsoids instead of spheres, or some inverse-skeletonization technique.

- **Thinness**:
  Curve-skeletons should be mono-dimensional, meaning that they should

be composed of only line segments in case of surface-based or point-cloud-based methods or have at most one-voxel thickness (according to the chosen neighbourhood) when volumetric methods are used.

- **Centeredness**:
  Ideally, a curve-skeleton of an object should be exactly centered within the object itself, hence it should be medial to the medial axis of the object, since the latter is guaranteed to be centered within the object. However, due to the sensitivity of the medial axis to small perturbations of the surface, exact centeredness it is not always required or even desirable.

- **Reliability**:
  An important property of a curve-skeleton is that every point on the object's surface should be visible from at least one curve-skeleton point, meaning they can be connected using a straight line completely internal to the object.

- **Junction Detection and Component-wise Differentiation**:
  The task of detecting the *logical components* of a shape is difficult to be defined as an algorithm, since it is more a semantical reasoning than a geometrical processing. A curve-skeleton should be able to encode the structure of a shape representing each component with a different branch, and identify where two or more different components meet, encoding this information as particularly relevant points. Usually it is possible to identify these points analyzing the skeletal structure, since they are the ones linked to more than two other points. As will be discussed in the following chapter, the ability to detect junction points varies greatly between the current state of the art extraction methods.

- **Connected**:
  This property is a consequence of homotopy, since in order to preserve the topology of a single connected object, the curve-skeleton must be a single connected component. However a connected curve skeleton could not be homotopic to the object it describes, i.e. not all the handles of the object are represented with a skeletal loop.

- **Robustness**:
  Discussing the centeredness of the curve-skeleton has been pointed out that if it is the exact centerline of the medial axis of a shape (exact centeredness), it will be affected by the same sensitivity to noise of the medial axis. A desirable property for the curve-skeleton is to be less influenced by small variations or noise on the object's surface than the medial axis.

- **Smoothness**:
  The requirement for a curve-skeleton to be smooth is not only an aesthetic property, but it becomes rather important in some applicative scenarios. For example if a curve-skeleton is used as a guide for virtual navigation applications (i.e. virtual endoscopy), smoothness is beneficial in order to have nice and fluid virtual camera movements.

- **Hierarchical structure**:
  Being the curve-skeleton an abstraction of a complex shape composed of different parts it should be able to encode the natural hierarchy of those parts, eventually being able to exploit its representation at different levels of detail. This property is particularly useful in shape matching scenarios and in animation contexts where, in order to obtain a plausible animation, the different parts of the shape move accordingly to their natural hierarchy.

Chapter 2

Structured Polygonal Meshes

The most commonly used representation for digital objects is the *Polygonal Mesh* structure. The main idea is to represent the object's surface using a set of simple polygonal cells, such as triangles or quadrilaterals (*quads*), assembled together. This kind of structure have shown to be versatile enough to efficiently store, display and process an object's surface geometry. According to the composition of the polygonal complex we can define a mesh as :

- **Triangle Mesh**: composed of triangles only.
- **Quad-Dominant Mesh**: mainly composed of quads, but it contains also of a small number of other polygons, typically triangles or pentagons.
- **Quad Mesh**: composed of quads only.
- **Polygonal Mesh**: composed of general polygons.

Historically, 3D models have always been represented using triangle meshes due to the nice properties that triangles can guarantee. For example, given a general triangle one can always count on the fact that they are always flat and convex. Moreover it is always possible to define linear interpolations on their vertices, and they are also easy to rasterize, in fact starting from the early days of computer graphics, the vast majority of hardware and software have been designed to efficiently work with triangles.

Quad meshes, however, have been widely used for long time in Computer Aided Design (CAD) applications and they currently represent a relevant

research topic and their advantages are well understood. One important advantage is that quad meshes are able, if carefully crafted, to exploit the higher level structure of an object, separating its meaningful regions and parts, in this sense it is possible to say that triangle meshes are **unstructured** while quad meshes can be **structured**.

This thesis is focussed showing how structured meshes can be obtained from unstructured ones or composed together, hence two kinds of structured meshes, Quad Meshes and Polar Annular Meshes, will be presented in the following sections.

2.1 Terminology

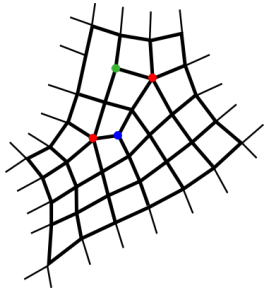
A polygonal mesh is described by its fundamental components, that are: *vertices, edges* and *faces*. Vertices are points in space, edges are line segments connecting pair of vertices and faces are polygons formed by closed chains of edges. Analyzing the meshes' connectivity we can infer some information, useful to categorize them. A mesh is said to be *manifold* when at each of its edges exactly two faces meet; this condition also imposes that the set of faces incident to a vertex is topologically equivalent to a disc. A *watertight* mesh represents a closed surface, which can be, intuitively, filled with water without leaks. Usually any two faces of a watertight mesh can meet on a single vertex or along a shared edge; when this property holds on the entire complex, the mesh it is said to be *conforming*. Special cases of *non-conforming* meshes are *T-Meshes* that contain the so called *T-vertices*. When an edge e of a face f coincides with the edges $e_1, e_2, \ldots, e_n$ of two or more faces adjacent to f along e, we call *T-joints* each vertex shared by each pair of edges (e_i , e_{i+1}).

A relevant property of a vertex is its *valence* which is the number of its incident edges.

2.2 Quad Meshes

The *valence* of a vertex, while being a small property, becomes of fundamental importance in the definition of structured quad meshes.

Since intuitively a quad mesh is composed of only quadrilateral faces, the simplest quad mesh is an imaginary infinite grid where each vertex has valence 4. The valence of a vertex is defined for each kind of mesh, but it becomes particularly important for quad meshes where vertices are called *regular* if they have valence 4 and *irregular* if their valence is different, typically 3 or 5. Irregular vertices are fundamental for quad meshes for at least



two reasons. The first one is purely practical: 3D artists have developed a common approach of managing the mesh topology of a shape, called *edge-loop modeling* in which edge loops are used to denote the boundaries of semantically meaningful parts of a shape or structure that will be animated. In this approach irregular vertices play a fundamental role, since only at their location the edge loops can change direction. In the inset image, irregular vertices are depicted respectively in red if valence is 5 and in blue if valence is 3; green denotes a T-vertex, while all the other vertices are regular. Discrete functions can also be defined on meshes associating values to each vertex or face of a mesh. As example a *cross-field* can be built, assigning a pair of orthogonal directions to each vertex and discover that the field singularities arise exactly on irregular vertices, which is why they are important also in a mathematical sense. Moreover considering all the shortest edge chains connecting irregular vertices brings to the definition of the so-called *Quad-Layout* which is a natural partition of the manifold and exploits its higher level structure. A common classification of quad meshes defines three classes of regularity:

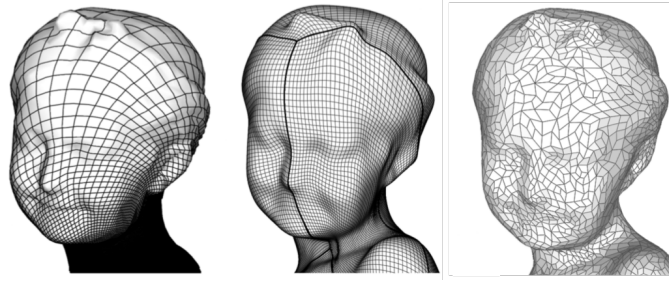


Figure 2.1: *A regular mesh (left), semi-regular mesh (center) and an unstructured one (right). Image courtesy of [BLP*13].*

- A *Regular mesh* contains only regular vertices as our infinite grid. As can be intuitively understood, its scope of usage is limited since it can be used only to represent closed surfaces with disc-like or torus-like topology.
- A quad mesh is said to be *Semi-Regular* if it is composed of a possibly small number of regular 2D *patches* glued together in a conforming mesh. A semi-regular mesh allows irregular vertices only on patch corners; their number and position on the surface along with the number and position of the patches represent a common criterion to assess the quality of a quad mesh. A note needs to be made about *valence semi-regular* quad meshes which are meshes where the majority of their vertices have valence 4. However if all semi-regular meshes

are also valence semi-regular, it is not true that every valence semi-regular mesh is semi-regular, since it is not guaranteed that they can be partitioned into a small number of patches. This distinction is also useful to understand that the number of irregular vertices alone is not a good quality indicator for a quad mesh.

- A quad mesh is said to be *unstructured* if a good part of its vertices are irregular preventing the mesh to be partitioned into a set of meaningful patches.

It is important to note that, except for the definition of regular meshes, the definition for the other classes gives no clear parameters to determine if a mesh belongs to that class or not.

Advantages of quad-meshes and their applications

There are a number of reasons why a quad-mesh can be preferred in many applications to other representations.

- On most shapes it is possible to detect two main local directions to which quads can be aligned. These directions are usually associated to principal curvature or sharp features and being able to align the mesh geometry to such directions enable to well capture the semantics of an object. This is particularly useful when an object will be segmented or animated. Triangle meshes do not allow such alignment, since they require to choose the direction of the third edge that does not reflect such alignment.

- Quad meshes are the preferred representation to be used as control meshes for high-order meshing and subdivision schemes.

- Patches of semi-regular quad meshes can immediately and efficiently be used for all kinds of mappings: textures, displacement, normals, etc.

Nowadays quad meshes are used in many common applications such as:

- **Modeling**: With the increasing popularity of digital sculpting the resolution of the objects being modeled have exploded and practitioners defined a common workflow which relies on sculpting an initial high-resolution version of the model, without putting any attention in obtaining a good topology, and then define a low resolution quad mesh with refined topology on top of the sculpted surface. This process is known as *retoplogy* among 3D artists and *remeshing* in the geometry processing community. Once this low-resolution mesh has been created, it can be used as control mesh for Catmull-Clark subdivision

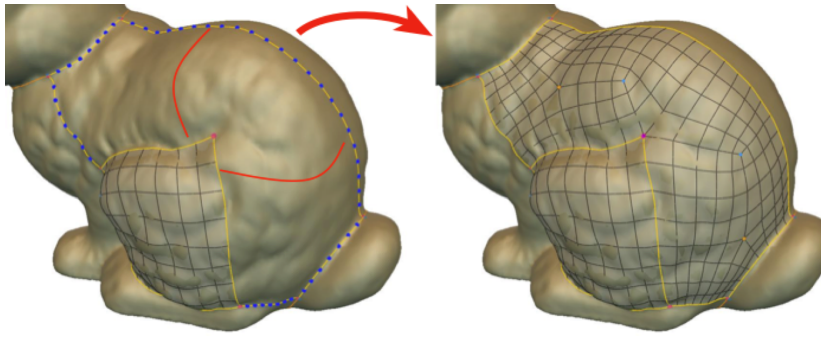


Figure 2.2: *A modern interactive retopology tool [MTP*15].*

surfaces producing a multi-resolution hierarchy. The details of the sculpted surface can be preserved also using displacement maps over the quadrilateral patches. Artists usually prefer to deal with quads instead of triangles or general polygons due to a number of advantages that quad meshes induce in their work. In quad mesh, if carefully structured, will have the edges aligned to the shape's features defining loops that are useful to define both smooth and sharp features. When a change in resolution resolution is needed, quad mesh subdivision is easier and give rise to results that are predictable and the overall geometrical structure can be retained. Quad meshes allow also to easily demarcate the boundaries between different parts of a shape, bringing to smaller distortions during animation, they also provide the ability to define domains for UV mapping in a more easier and intuitive way.

- **High-order surface modeling**: Semi-regular quad meshes can be used as control meshes for subdivision surfaces, moreover they are perfect also for fitting splines or NURBS which are *de facto* standards in some industrial applications; in fact each spline patch is defined for each patch (or *domain*) of the quad mesh and then glued together at shared boundaries. They have shown to be much easier to handle and manipulate than triangle-meshes in these applications.

- **Texturing** is a good example for showing why quad-meshes can be easier to manipulate than other representations, since each patch of a semi-regular quad mesh can be easily mapped to a rectangular texture allowing for mapping images, displacements and normals in order to add low frequency details to low-resolution meshes.

- **Compression**: High-resolution 3D models can be stored as coarse quad meshes (or quad-based higher order surface), within displacement mappings to the quadrilateral domains.

- **Finite Element Analysis**: The Finite Element Method (FEM) is a mathematical tool that allows to approximate solutions to complex problems expressed as partial differential equations with an associated set of constraints, called *boundary conditions*. It has a variety of applications, especially in mechanical engineering, allowing run a number of physical simulations. FEM can be applied to meshes represented with bidimensional (triangles, quads) or tridimensional (tetrahedra, hexahedra) elements. Quadrilaterals and hexahedra are preferable to triangles and tetraheadra, since they are able to produce more accurate results more efficiently. Some types of FEMs benefit from having the elements well aligned to the boundary geometry or the directionality of the physical phenomena being simulated.

- **Semantic Segmentation** A problem that arises when segmenting a shape is the complexity of obtaining semantically meaningful segmentations using solely geometrical data. Quad meshes helps to cope with this problem due to their natural alignment to shape's features, indeed the geometry of a well structured quad mesh provides powerful indicators for a semantical shape segmentation. Recalling how 3D artists take advantage of the edge loops in quad meshes, it appears natural to use the edge loops connecting the irregular vertices to extract a higher level and semantic information from the shape.

Quality estimation

Not all quad meshes are the same. As for triangle meshes, also for quad meshes is possible to define local and global quality indicators. Locally is desirable to determine the quality of each quad, while globally is desirable to investigate how these quads are arranged together.

Quadrilaterals in a 3D space are more complicated to handle than triangles, for example they are not guaranteed to be always planar, and planar quads can be non-convex. If it is possible to evaluate the quality of a triangle just looking at its largest angle and determining how much is similar to a equilateral triangle, for quads it is necessari to take into account more aspects.

It is desirable for quads to be as flat as possible and as squared as possible, having internal angles close to 90 degrees and opposite sides almost equal; these characteristics become crucial for some applications such as architectural design and physical simulations. From a global point of view, the aim is to have a good *edge flow*, meaning that the quads should to be arranged in order to be aligned with the shape's features, such as sharp creases of mechanical objects or principal curvature directions of tubular regions. Another example of feature lines that are aimed to be defined with the quad arrangement are Langer's lines of skin tension, that can be traced

on a human body, which correspond to the natural orientation of collagen fibers, allowing a human model to be animated realistically.

In order to better approximate the surface of an object and avoid to use a disproportioned number of quads, it would be preferable that the tessellation adapts its resolution, using a polygon density inversely proportional to size of the details on the surface. Large, more or less planar regions can be approximated with a few quads, while small scale features will need a higher number of smaller quads to be faithfully represented. This *resolution adaptivity* is achievable in two ways: adding more irregular vertices on the surface, which turns to have more patches, or allowing our mesh to be a *T-mesh*. Another possibility to adapt the resolution of our surface is to drop the requirement of having squared quads and allow for anisotropic quads which fill the regular patches with different resolution on the two directions. For some applications, such as intrinsically anisotropic physical simulations, this kind of tessellations are also desirable.

It is difficult to define a set of precise quality parameters for a quad mesh. It is known that is preferable to have a low number of irregular vertices, but at the same time there is no threshold or formula able to tell if more irregular vertices are needed or some of them are unnecessary.

Similarly it is desirable for the irregular vertices of a quad mesh to have a good placement and define a nice quad layout. However these desiderata tend to be more semantical than geometrical. Taken a human body modeled by a professional 3D artist, can be observed that irregular vertices are carefully placed where the shape should be animated, and the face of the model will have an higher density of extraordinary vertices than every other part of the body. Follows that some geometrical information can be used, such as principal curvature direction fields, as an indicator of where to place these vertices, but geometry alone can be not enough, since most of the information needed to carefully place irregular vertices is semantical, at least for animation characters. Part III will present a method for extracting a coarse quad layout from a triangle mesh; such method is able to use semantical information on the shape, conveyed by curve-skeleton. To further deepen the discussion about quad meshes please refer to the extensive survet provided by Bommes and colleagues in [BLP*13].

2.3 Polar Annular Meshes

In the previous sections a categorization of polygonal meshes, depending on the kind of polygonal cells used, was introduced. Moreover has been

said that, under some conditions, quad meshes can exploit the semantical structure of the shape, while triangle meshes are not able to do that. Nothing was said about the possibility of having **structured** *Quad-Dominant Meshes.*

Polar-Annular Meshes (PAM) [BAS14] are structured quad-dominant meshes which embed, within the mesh geometry, the skeletal structure of the shape. A PAM is composed of only two kinds of partitions :

- **Annular Regions** which are rings of quads.
- **Polar caps** which are triangle fans

The PAM structure takes inspiration from a previous work [BJD*12] called RigMesh. In the common professional pipelines modeling and definition of the kinematic-skeleton of a character are separate steps. In [BAS14] and especially Rigmesh [BJD*12] (from which it took inspiration) are exploited the benefits of being able to *co-model* both the shape and its skeleton operating solely on the mesh' geometry. The skeleton encoded into a PAM is able to provide semantical information about the shape and it is defined by the shape structure itself since each boundary of an annular region defines a joint of the skeletal structure.

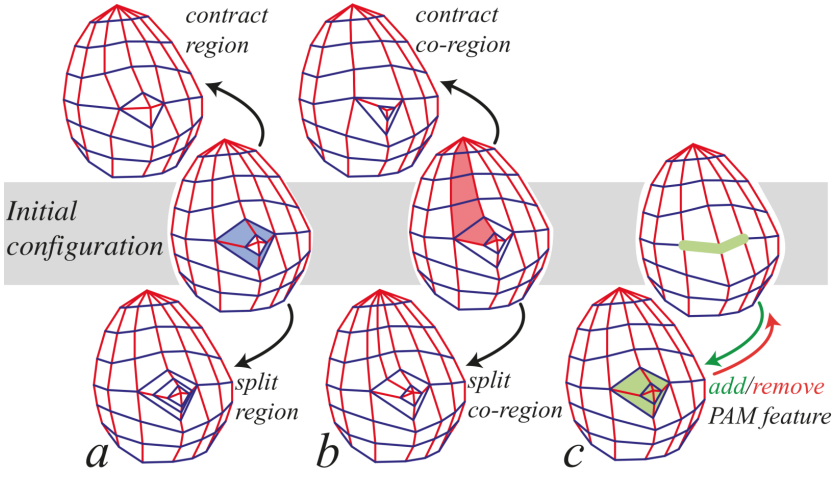


Figure 2.3: *PAM primitive operations: (a) and (b) show refinement and simplification of both regions and co-regions, while (c) shows how skeletally supported PAM features can be added to the shape. Annular regions are bounded by two loops of rib edges (depicted in blue). Image courtesy of [BAS14].*

On a PAM there are always two distinct kinds of edges: *spine* and *rib* edges, which respectively are the ones aligned with the local skeleton direction (red in Figure 2.3) and the ones that define the quad rings (blue

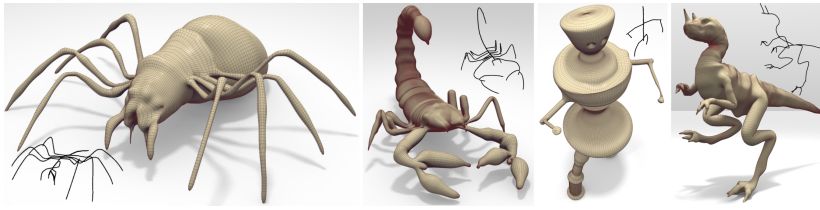


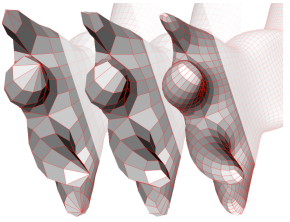
Figure 2.4: *Examples PAMs, and their implicitly defined skeleton, built using the iPad app provided by the authors. Image courtesy of [BAS14].*

in Figure 2.3). Chains of spine edges always connect pairs of poles, except for shapes like toroids, for which the PAM definition is ambiguous and each edge can be either a spine or a rib one. Rib edges always form loops orthogonally to the skeletal local direction; these loops are said to be *complex* where three or more regions meet. The skeleton of a PAM is implicitly defined, but it is always possible to reify it simply contracting all rib edge loops to their centroid in order to obtain the nodes. A pair of distinct nodes N_i and N_j will be adjacent in the skeleton if spine edges connecting the two rib edge loops that originated the nodes exist. Poles represent the particular cases since they can be seen as a rib edge loop consisting of a single vertex and they are always the terminal nodes of skeletal branches.

In [BAS14] a PAM is defined as a polygonal mesh consisting of only triangles and/or quads (refer to Figure 2.4) such that :

- each quad belongs to a single quad ring called *Annular Region*
- each triangle belongs to a single fan of triangles denoted as *Polar Region*
- the set of faces incident to a vertex is always homeomorphic to a disc
- no t-joints are allowed

PAMs naturally expose a set of primitive operations (Fig.2.3) that can be performed such as refinement or simplification of annular regions or quad stripes belonging to different regions adding or removing a chain of spine edges (depicted in red in figure 2.3). A set of quads bounded by two chains of spine edges are denoted with the term *co-region*. A PAM also allows to easily add and remove skeletally supported features; this can be done splitting a chain of rib edges creating a boundary loop, and creating a triangle fan along the vertices of this boundary. Connecting together two PAM features is also simple, it is necessary only to remove the polar caps and bridge the two boundaries with a supplementary ring of quads.



If the two boundaries have different number of edges it is necessary to refine or simplify one of them in order to match, this can however be a non-trivial operation in case of higher genus objects. Given the peculiar structure of the PAM, both subdivision and simplification operations must be performed with particular care, since it is necessary to preserve its unique properties. Authors of [BAS14] have implemented an extension of the *factored* Catmull-Clark that uses only local information and is able preserve the PAM structure. Simplification is done removing co-regions using a simple selection policy. Since the PAM structure could be unsuitable for other applications a procedure for quadrangulate the polar regions is provided, this can produce a quad mesh in almost any case. In order to always have a quad mesh a step of Catmull-Clark subdivision is required, hover the procedure do not aim to produce also a nice quad-layout, hence in some way the semantical structure of the shape is not preserved.

Since a PAM encodes the skeletal structure of the shape and its induced segmentation it can be posed, animated, cloned and transformed easily just operating on the geometry of the shape.

Part II

Interactive Curve-Skeleton Manipulation

In the first part of this thesis the concept of curve-skeletons has been introduced. Curve-Skeletons are convenient shape descriptors commonly used in a number of applications. Methods for automatic curve-skeleton extraction has been also presented. While developing the quad layout computation, that will be presented in Part III, which relies on the semantical information conveyed by the curve-skeleton, has been found that using automatically extracted skeletons often brought to sub-optimal results.

The reasons behind this non-optimality are mainly related to the fact that not having a clear definition of a curve-skeleton is, the existing skeletonization algorithms and approaches work in different ways, starting from different assumptions, leading then to results that may greatly differ in terms of the skeleton's structure.

From a practical point of view all the presented skeletonization methods have specific properties and issues, making it difficult to use the resulting skeleton without processing. Moreover, each of the presented methods require some parameter setting in order to regulate the algorithm's ability to small scale features.

One possible solution can be to write an algorithm able to process a curve-skeleton extracted with a particular algorithm and make it fit the specific needs. However, there are several issues related to this approach:

- Different skeletonization algorithms may give different results for the same shape. It is usually a good choice to extract the skeleton with different algorithms and choose the one that is best suited for the current application. Moreover computing a curve-skeleton that is optimal for a specific application is not a *one-click operation* since it requires a fine-tuned parameter setting that is potentially tightly coupled to both the specific shape and the specific applicative context.

- Most of the parameter setting of skeletonization algorithms is about detecting small scale features. However an increased sensitivity can derange the skeletonization process representing noise or small perturbations of the surfaces as spurious branches; on the other hand if a user wants to reduce the effects of noise or small perturbations she will also face the risk of not detecting all the meaningful features of the shape being processed. Furthermore it is difficult from an application-agnostic point of view, to define what is a meaningful feature and how much noise can be tolerated in order to have it represented by the skeleton.

- The desired features of the curve-skeletons that are going to be used in a novel pipeline are not always known a priori, but can be discovered during the development of the new technique.

- The information conveyed by a curve-skeleton is both topological and semantical. While the topological part can be algorithmically manip-

ulated, writing an algorithm able to process the semantical one can be really challenging.

A representative example, showing the different results of different skeletonization approaches on a rather simple shape, can be seen in figure 2.5. While the three skeletons have the same terminal nodes, their overall structure is different and it can be difficult to state which one is correct.

However rather than deciding if a skeleton is correct or not, it is more appropriate to evaluate if an extracted curve-skeleton is suitable and brings optimal results or not, in the specific application in which it will be used.

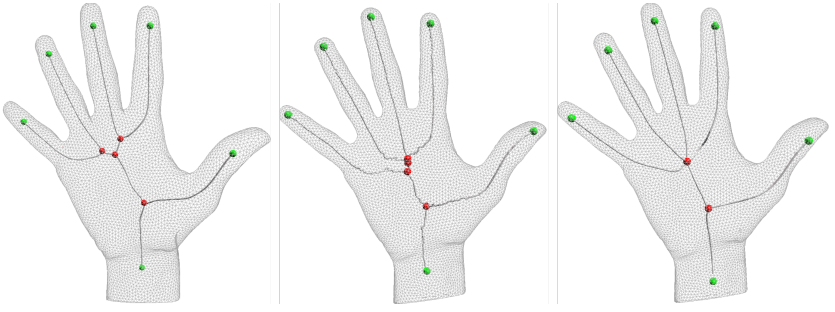


Figure 2.5: *A direct comparison of three automatic skeletonization methods. [TAOZ12] (left), [DS06] (center) and [LS13] (right) on the same hand model. .*

For a human it is easy to note that at an higher level, the three skeletons are rather similar and can be easy to manipulate them in order to morph each of the three into one of the others. The same task can be difficult to encode into an algorithm able to work on a general shape's skeleton.

What has been observed is that it can be easy and practical to rely on a pipeline which involves a skeletonization algorithm with standard parameter setting, followed by a manual editing stage with an interactive tool. Especially for researchers, when they need that need a curve-skeleton as part of the input of a newly defined or existing pipeline, having an interactive tool for quick editing or repairing the skeletons, is a useful addition to their toolbox.

Following chapters will present *Skeleton Lab* [BMUS15], a interactive tool for creating, editing and repairing curve-skeletons, that was presented at the conference STAG (Smart Tools and Applications in computer Graphics) 2015 in Verona (Italy) and awarded as Best Paper of the conference and invited to submit an extended version to Computer&Graphics.

Related works

All the methods and software for inverse skeletonization presented in Section 1.4 and in Section 1.3 comprise a basic interactive skeleton creation tool. Since the aim of these tools is to quickly create meshes that need to be refined, especially using a digital sculpting approach, they allow the user to perform only simple operations on the skeleton, such as:

- Add/remove a node.
- Change the position of a node.
- Change the radius of a node.

These basic operations do not 123 providea sufficiently wide set of operations for a user, especially a researcher, that needs to easily process a skeleton that has been 125 computed with one of the skeletonization methods presented in Section 1.2.

Skeleton Lab addresses the need of easily editing an automatically extracted skeleton at different granularities, allowing for both geometrical and topological operations, in order to obtain, within minutes, curve-skeletons that are tailored for the user's specific tasks. The great part of the provided operations are not present in other tools and have shown to be very useful in practice.

Chapter 3

Skeleton Lab

The tool that has been developed allows the user to handcraft new skeletons from a scratch or to load and edit existing ones. Given this twofold purpose it provides some basic operations, that are present in all the tools discussed in previous section, along with a set of functionalities that the other skeleton editors lack.

In Skeleton Lab the skeleton is represented as an attributed graph $G = (N, L)$ where N is the set of the nodes and L the set of the links between the nodes. Each node $N_i \in N$ has a few attributes :

- Position in $\mathcal{R}^3$.
- Radius of the associated maximal sphere.
- Type of the node. Follows classification.
- List of the nodes linked to it, since the links are not explicitly stored.

Node Types

The nodes of the skeleton are classified, mainly depending on the number of neighbours, as *Joint nodes* (**Jn**) that have two linked neighbours; *Leaf nodes* (**Jn**) that have only one linked neighbour; and *Branching nodes* (**Bn**) that have only three or more linked neighbour. In Skeleton Lab a user is also allow to mark a **Jn** as an *Articulation* (**An**), that is a *Joint node* that is semantically important (e.g. a human ankle or elbow).

A sequence of linked nodes starting and ending with either a **Bn** or a **Ln** is a *Branch*. When a branch ends with a **Ln** it is called **Ln**-*ending*, while a *Looping-Branch* is a branch that starts and ends with the same node. These terms will be used throughout the thesis when referring to the nodes of a skeleton.

Skeleton Lab allows the user to visualize and manipulate a curve-skeleton and optionally visualize the triangle mesh that it is connected to. The user can also create a curve-skeleton from a scratch, with or without having a mesh as reference. From a design point of view, the functionalities of the tool are grouped depending on the granularity on which they operate.

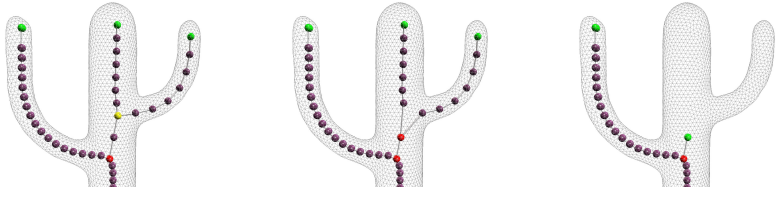
There are three modes:

- **Node mode** in which it is possible to operate on a single node or a set of selected nodes.
- **Branch mode** which allows to directly manipulate an entire selected branch.
- **Skeleton wide** which work on the entire skeletal structure.

Manual editing is an iterative process, which often involves undoing part of the work that has been done, in order to make corrections or to change previous choices. Moreover, have been observed that editing a curve-skeleton in order to make it fit a specific pipeline requires a number of attempts before getting it right, hence a full undo/redo stack has been necessarily implemented in the presented tool.

Basic Operations

The tools discussed in Section II allow the user to perform just basic operations, such as : adding a new node connected to an existing one, translate and rotate in space a selected set of nodes, change the radius of their maximal sphere. All these operations, being necessary, are also present in this tool and do not need further explanation.



(a) **(b)** **(c)**

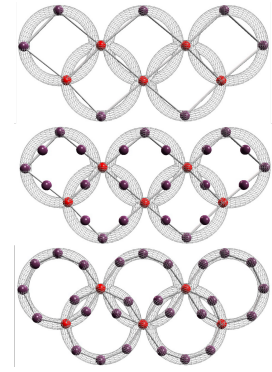
Figure 3.1: *a) a skeleton with selected the* **Bn** *that is going to be removed. b)* **Bn** *removed and all its links transferred to a neighbor; c)* **Bn** *removed and all its incoming* **Ln***-ending branches removed.*

3.1 Node Mode

In *Node Mode* a user is allowed to operate at the highest degree of granularity, that is a single node or a selection of nodes. The following sections will present the capabilities of the tool with respect to this aspect.

Add Midpoint and constrained movement

An efficient approach for handcrafting a curve-skeleton is to work in a coarse-to-fine approach, where the general structure of the skeleton is sketched at first, and it is subsequently refined adding new nodes and carefully position them. According to this approach the tool allows to select two connected nodes and add a new node, connected to both, positioned at the midpoint of the two. In order to fine tune the position of the nodes, it is possible to constrain their movement to positions that are linear interpolation of its neighbours. Given the nature of this constrain, this can be applied only to **Jn**'s, since they have exactly two neighbours.



Remove a node

Removing a node is a very basic operation, but it can have different outcomes depending on the type of node being removed. The easier case is represented by the **Ln**'s, since they can be removed without any further operation. Also the **Jn** case is trivial, being that it can be removed safely, just recovering the original topology, connecting together the two neighbours of the removed node. In the case of removing a **Bn**, the user is allowed to choose between two possible outcomes of the removal, visually explained in figure 3.1. The first possibility is to remove all the links of the node, which means that all the **Ln**-ending branches connected to the selected **Bn** will be removed. The other branches connected to the **Bn** being removed will be maintained, and will undergo a pruning since one of their terminal nodes will be removed. The second possibility is to keep the original structure transferring all the links of the **Bn** being deleted to one of its neighbours selected by the user.

Copy and Paste

Especially when a user wants to create a skeleton from a scratch, it may happen that she wants to replicate a part of the skeleton that she is editing.

Skeleton Lab allows to copy part of the skeleton and paste it elsewhere; in order to do that the user is required to select a set of connected nodes, select one of them as *source* node for the copy and the *destination* node with which the *source* node will be merged in order to perform the paste operation. The *destination* node will inherit all *source* node's copied neighbours. It is noteworthy that copy and paste are disjoint operations, meaning that one can copy then and perform the paste operation after some skeleton modifications.

Explicit topology management

Since a curve-skeleton encodes also topological information about the shape and it is represented as a graph, the user is allowed to explicitly manipulate the skeleton's topology in two possible ways:

- Merge two not connected nodes creating a looping branch.
- Break an existing link between two selected nodes.

As design choice, the user is not allowed to explicitly create multiple connected components, since as stated by Cornea in [CSM07] the curve-skeleton of single objects should always be a single connected component. Due to this choice, when the user tries to break a link between two nodes, Dijkstra's algorithm is run in order to check if the skeleton will still remain a single connected component. However, since it may happen that an automatically extracted curve-skeleton is not a single connected component, the tool is able to manage them in order to allow the user to fix them.

3.2 Branch Mode

The operations described so far define can be performed on just a single node or a selection of nodes; here are present a set of operations that can be applied on a whole skeletal branch.

Pruning

Pruning provides a quick way to manipulate **Ln**-ending branches, allowing to delete the **Ln** of the selected branch. This has the consequence of shorten the branch by one node, without changing is structure. If the branch on which the pruning operation is applied is composed of a single node, the result is the same of a branch collapse.

Branch collapse

A common issue of skeletonization algorithms is the presence of spurious branches, that are skeletal branches that do not reflect any meaningful feature of the shape, or worst they define a skeletal topology that is different from the real one. Recovering this situation can be non-trivial to encode into an algorithm, generic enough to handle the wide variety of cases that can arise on different shapes.

Usually, it is easier for an human operator to detect these situations and decide which should have been the right topology and the operations needed to obtain it starting from any skeleton of any shape. For these reasons, the user it is allowed to remove branches with a single operation. If the branch is an **Ln**-ending one, it will be removed without further operations, if it is an internal one, that a branch with both the endopints that are **Bn**'s, they will be merged into one after removing all the other nodes. In case of a looping branch all its **Jn**'s are removed.

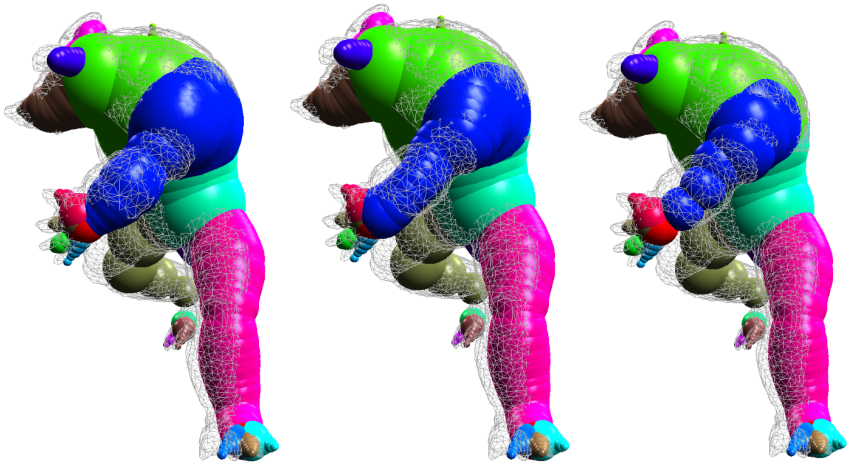
Resampling

Skeletons computed with state-of-the-art skeletonization algorithms are often composed by a large number of nodes, in fact they can easily reach a thousand nodes, even for models of moderate complexity. In usual applications, this level of detail leads to redundancy and is not necessary, indeed it could be preferable to deal with a smaller number of nodes. They are usually too dense to be used as they are, hence they often need to be simplified. In order to address this issue in a practical way, the user is allowed to resample the skeleton. Resampling is done useing arc-length parameterization with a slightly modified version of the method presented in [Hec94], in which both position and radii of the nodes is involved. After this operation, the 3D structure of the branch is kept unchanged while position and radii of the new nodes are approximated from the original ones. It is noteworthy that super-sampling is a smoothness preserving operation, while sub-sampling is not.

As presented in Section 3 some **Jn**'s can be marked as **An**'s so that they become semantically meaningful nodes. Due to their special nature, each branch after resampling must have, at least N_min nodes where :

$$N_{min} = \begin{cases} 1 & \text{if the branch is a looping one} \\ E + \#\textbf{An} & \text{else} \end{cases}$$

where E is the number of distinct endpoints of the branch. This operation allows to quickly change the skeleton resolution adaptively to the need of each application and each branch of the skeleton.



(a) (b) (c)

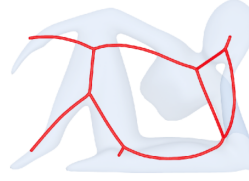
Figure 3.2: *a) Armadillo model with its curve-skeleton shown with the associated maximal spheres. b) The branch highlighted in blue and representing the arm was downsampled from 36 to 20 nodes c) The same branch downsampled to 8 nodes.*

3.3 Skeleton Wide Operations

Along with the previously presented operations, Skeleton Lab offers also a set of operations which work on the overall structure of the loaded skeleton in conjunction with the mesh to which it refers. In order to perform such operations, a spatial indexing data structure is needed to be able to perform ray casting operations and distance queries. A number of such data structures exist, Octrees, OOB's, kD-trees and so on, however the choice has been to rely on AABB trees which provide good performances and a very functional implementation can be found in the publicly available CGAL library [CGA15]. In order to build the AABB tree of the selected mesh, this mesh needs to be triangulated first, however this can be done in very trivial ways.

Fix nodes outside of the mesh

Most skeletonization algorithms cannot guarantee that their output consists solely of nodes that are on the interior of the input shape. Even the most robust ones, such as [TAOZ12] depending on the parameter values chosen, can lead to branches that cross the mesh surface, instead of flow inside it.



The property of being internal to the shape can
be crucial in some applications, such as remeshing or mesh reconstruction, hence this can be a very stringent requirement. In order to solve this issue a simple but effective procedure can be performed.

For each node N_i its closest mesh face F is found, and its normal N_F is computed. The position of N_i with respect to the mesh is checked, if it lies on F or it is located in the negative half-space defined by the supporting plane of F it is inside of the mesh, otherwise it is outside and its position needs to be changed. If N_i is outside the mesh its new position is computed as

$$N_i - [d(N_i, F) + \epsilon] * N_F$$

where d is the Euclidean distance and ϵ a small positive number.

Approximated re-centering

As stated in [CSM07] a desired property for curve-skeletons is their centeredness, which is the property of being medial to the shape it describes. Cornea and colleagues state that *perfect centeredness* is obtained when the curve-skeleton lies on the Medial Surface, following the 3D extension of Blum's Medial Axis), and it is centered with respect to it. However this property can be difficult to obtain and sometimes undesirable due to the well known sensitivity of the medial axis to noise and high-frequency details of the surface. Moreover, achieving perfect centeredness can be difficult to obtain both during skeleton extraction and as post-processing since most state-of-the-art methods do not provide the medial axis as output and some methods do not rely on medial axis computation to extract the curve-skeleton.

Starting from the description of *approximate centeredness* given in [CSM07] the following algorithm has been implemented.

For each skeleton's node N_i the term $\overrightarrow{N_i}$ denotes the local direction of the skeleton computed as :

$$\overrightarrow{N_i} = \begin{cases} \overrightarrow{N_j N_i} & \text{if } N_i \text{ is } \textbf{Ln} \\ \overrightarrow{N_i N_k} & \text{if } N_i \text{ is } \textbf{Bn} \\ \overrightarrow{N_j N_i} + \overrightarrow{N_i N_k} & \text{if } N_i \text{ is } \textbf{Jn} \end{cases}$$

where N_j and N_k are, respectively, the predecessor and the successor of N_i in the branch they belong to.

As can be seen in Figure 3.3, a set of n uniformly distributed radial rays with origin in N_i and orthogonal to $\overrightarrow{N_i}$ are cast and their intersections with the object surface are computed. For each ray R_h only the intersections obtained pinching the object from the inside is considered and the intersection I_h that is nearest to N_i is stored. A pair of opposite rays is discarded if at least one of the intersections is not valid.

For each pair of opposite rays $(R_h, R_h^{'})$, C_h is computed as the midpoint of their intersections $(I_h, I_h^{'})$, then the new position of N_i is computed as the centroid of all the C_h's and the new maximal ball radius as the average semi-distance of all the intersection pairs. If N_i is a **Bn** its new position and radius are computed, respectively, as the centroid of the positions and mean radius calculated separately for each incident branch.

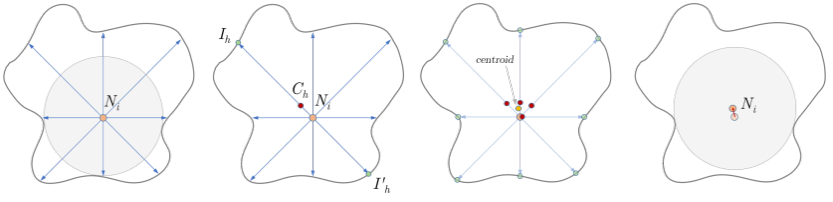


Figure 3.3: *A section of the tubular branch where the node N_i is centered. On the left the maximal ball and some of the rays casted are casted; on the middle left is shown how to compute the midpoints; on the middle right is shown how to compute the centroid; on the right the new position of N_i and the new maximal ball are shown.*

The presented procedure works quite well even if its results are dependent to the original node's position and the skeleton directions. Has been observed that a single iteration of the procedure is usually not enough to obtain optimal results, however since convergence cannot be guaranteed, it is up to the user to iterate the procedure until the results are satisfactory.

Chapter 4

Discussion

The presented work have been developed as a response to practical issues related to the use of curve-skeletons as part of a processing pipeline. It has been shown that different skeletonization algorithms have different peculiarities; although some may be more robust than others, there are no theoretical nor practical guarantees that automatically extracted curve-skeletons can be used as they are in any processing pipeline. This is not to be intended as a defect but as a natural consequence of the variety of approaches and hypothesis behind each approach.

What has been observed is that can be more practical to use an interactive tool as Skeleton Lab to quickly edit or repair curve-skeletons in order to make them fit a prescribed pipeline, instead of using the automatically extracted ones as they are, or spending time and efforts to build algorithms that process these skeletons in order to make them reflect some desired property. Especially the latter can be particularly challenging due to the semantical nature of the information conveyed by the curve-skeleton.

Another key aspect that makes Skeleton Lab a practical addition to a researcher's toolbox is that curve-skeletons, while being one of the most broadly used shape descriptors, are still lacking a precise, mathematically sound and universally accepted formal definition, hence a direct manipulation can be seen as a safe operation which is allowed to take into account only the final results that a user wants to accomplish.

4.1 Results

In this section examples of interactive modifications to automatically extracted curve-skeletons will be shown. The basis of these results are the shapes used as input in the pipeline of the method presented in [Usa15]. Figure 4.1 shows one of the most challenging skeletons. Due to its complicated

structure the *Dragon* model is challenging also for state-of-the-art skeletonization methods; in fact it was possible to compute the curve-skeleton only using [DS06], however it resulted in different connected components with a number of disconnected branches, failing to detect the head structure. In order to obtain a usable skeletal structure the first step was to reconnect all the separate branches recovering the head's connectivity, then it was subsampled and some nodes where moved to better represent the shape's structure.

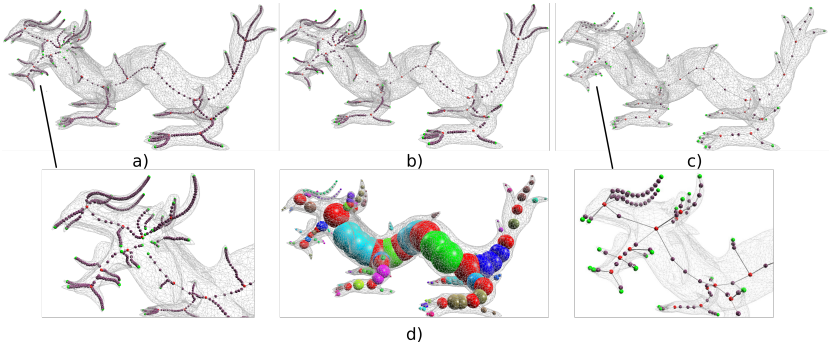


Figure 4.1: *Editing process of a complex skeleton extracted with [DS06] (a, b, c). In the second row the skeleton is rendered with its associated maximal balls. Close ups show the defective topology of the dragon head (left) and how it was fixed (right).*

The entire editing session required about 15 minutes for a trained user. Another example showing the fact that some skeletonization methods, while producing correct results, may fail to convey the correct semantical information about the structure of a shape can be seen on Figure 4.2 where the octopus mesh and its skeleton are showed.

According to the definition given by the Oxford Dictionary of English an octopus is a *a mollusc with eight arms and a soft body.* Moreover, according to this definition, its usual shape and its typical representations one can define its shape as a body with 8 tentacles starting from the its lower part. However, the curve-skeleton extracted using [TAOZ12] presented a number of spurious branches which should not be part of the skeletal structure. To recover the structure with a single **Bn** and 9 branches (right part of Figure 4.2) that a human would expect, a 4-minutes session has been necessary.

4.2 Applicative Contexts

Skeleton Lab was conceived during the development of [Usa15] since has been observed that using the results of skeletonization algorithms without

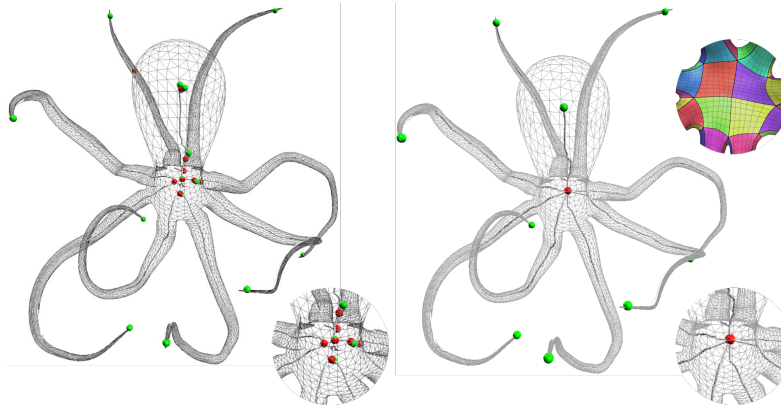


Figure 4.2: *Skeleton of an octopus model extracted using [TAOZ12] (left) and its edited version (right) with all the spurious branches collapsed. Close-ups on right side of the image show how the unnecessary* **Bn** *'s were removed in order to obtain the quad layout depicted on the top right.*

further processing often not allowed to obtain optimal results from the pipeline. It was observed also that :

- Optimal skeletons for that application are robust, connected and reliable. Strong homotopy was not crucial, but beneficial.
- The pipeline had better performances and results when the number of nodes was small, but the skeleton was still well approximating the shape.

Once Skeleton Lab was developed, it was used both to investigate the desirable properties of the used skeletons and to perform interactive refinements. In some cases, optimal skeletons for this application have been obtained with little interactive editing of the automatically extracted curve-skeletons, collapsing spurious branches, handcrafting the missing ones and subsampling part of the branches.

Resampling have shown to be one of the most useful operations due to the high resolution of the results of skeletonization methods, which are composed of thousands of nodes even for shapes of moderate complexity. Curve-skeletons with small number of nodes are beneficial also in other application fields, such as animation, where kinematic skeletons are composed of very few nodes for each bone.

In this application an automatic skeleton simplification may lead to results that are not well suited for the purpose. The position of the nodes of kinematic skeletons is crucial since they are associated with articulations of the shape that need to be preserved, but algorithmically capture if a skeletal

node is an articulation or not is a semantical reasoning that is difficult to express in an algorithmic fashion.

Part III

Quad Layout extraction

Recent years have seen a widespread diffusion of a novel modeling metaphor called *Virtual Sculpting* which mimics clay-based sculpting techniques. The first software to adopt this paradigm was ZBrush [Pix07] and it was soon followed by other competitors such as MudBox [Aut07] and 3D Coat [Pil09]. For free-form shapes or character modeling, virtual sculpting provides a more flexible and intuitive metaphor than previous modeling techniques, such as NURBS or polygonal modeling. This approach has the advantage of being closer to a creative process than other techniques, more focussed on CAD applications.

As popular adage says *There is no such thing as a free lunch*, hence virtual modeling too, despite it frees and supports the artist's creativity, comes with some drawbacks. From a technical point of view the main disadvantage is the irregularity of the dense triangle meshes produced by such systems, which have similarities to the results of range scanners. Target models for the modeling and animation pipelines, are instead usually required to be surfaces defined over a coarse control mesh of quadrilaterals, as in the case of subdivision surfaces.

Obtaining such control mesh from a dense and unstructured triangle mesh is a challenging problem, known as retopology or remeshing, depending on the audience. This problem is challenging due to the different aspects which compose the quality metric for such control meshes, which can be synthesized as coarseness and geometric fidelity [CBK12].

Different approaches to automatize this process have been proposed, however, while the results are encouraging, the problem is still open and the current methods are not reliable enough to be used in real-world scenarios. There are methods able to convert triangle meshes to quad-only or quad dominant meshes, however they are still unable to capture the overall structure of the shapes, that is required in various contexts such as deformation and animation. Commercial modeling tools offer their own retopology methods, however they still require a supervision and/or substantial guidance from a professional user, whose efforts to build the structured version of a sculpted model, can sometimes become comparable to the efforts required to create it the first time. Conversely, a common workflow used with traditional polygonal modeling tools relies on iterative subdivision, extrusion and refinement of the current shape, starting from a cube. Sometimes different parts of the final object are modeled separately and then stitched together. This process, compared with virtual sculpting, is less artistic, more tedious and time consuming, however it produces, by direct construction, a control mesh of pure quads that naturally aligns to the structure of the intended shape being modeled.

The following chapters will present a novel and automatic method for producing a coarse quad layout of a digital objects, taking as input a triangle mesh and its curve skeleton. The quad layouts obtained with this approach contain few irregular vertices of low valence, fewer patches than

previous methods and follow the overall structure of the objects, thanks to the topological information provided by the curve-skeleton. This work has been accepted for publication to ACM Transaction on Graphics [Usa15] and presented at the SIGGRAPH Asia conference in November 2015.

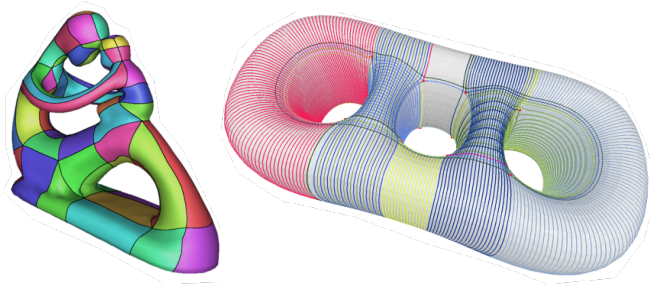
Problem definition

Given a surface mesh $\mathcal{M}$ build a *quad layout* $\mathcal{Q}$ that is a partitioning of $\mathcal{M}$ into non-overlapping quadrilateral *patches*, represented by its vertices, edges and faces as an extremely coarse quad mesh. Estimating the quality of a quad layout is a difficult task due to fact that it is application-dependent and, to some extent, semantically-dependent. However, can be defined set of desired properties that should be taken into account when evaluating a quad layout's quality, which is similar and partially overlaps with the the quality criteria given for quad meshes. The problem of producing such quad layouts has been studied in different previous works, and the following considerations about the goals of a quad layout computation technique, try to summarize observations and results presented in relevant previous works [CBK12, BCE*13, BLP*13, CK14a].

- **Patches quality**. One of the ideal requirements for a quad mesh is to have its elements, the quads, as close as possible to planar squares. In case of quad layouts, where each patch can span for a wide area of the input surface, this is, in practice, not achievable nor desirable. Patches defined on the surface are quadrilaterals only in a topological sense, meaning that there are no guarantees that their shape resembles a quadrilateral (Figure 4.3.a), therefore we would like the patches to resemble a quadrilateral having its corner angles close to $\pi/2$ and their edges close to geodesics over the surface.

- **Irregular vertices**. An important property of semi-regular quad meshes, thus of quad layouts, is the number and placement of their irregular vertices. While it is desirable to keep their number low, they are necessary in order to manage the mesh' edge flow and enable adaptive meshing. Irregular vertices are so important, that 3D artists have developed a common workflow for topology management in order to obtain quad meshes that animate properly. This workflow is often called *edge loop modeling* and prescribes to use *separatrices* to demarcate the boundaries of structures that will be animated and obtain a good edge flow. Most of this work involves optimizing the location of irregular vertices. To some extent it is possible to state that for the purpose of animation, the location of irregular vertices is more important than their number. Another key aspect is the valence of these

vertices which is usually low, typically 3, 5 and 6, for user-defined topologies.



(a) **(b)**

Figure 4.3: *a) Some of the patches over the fertility model are not well shaped, especially the green one at the center. b) Particularly entangled singularity graph connecting only 16 irregular vertices, originating thousands of domains. Image (b) courtesy of [TPP* 11].*

- **Patch structure**. A low number and good placement of irregular vertices alone is not enough to guarantee that a quad layout (or quad mesh) has a good structure. The connectivity between these vertices is also very important. What CG artist do with their *edge loop modeling* is defining at the same time both irregular vertices and the connectivity between them, aligning separatrices to meaningful shape features. As will be discussed in the state of the art review, some automatic methods for quad remeshing or quad layout computation, separate the two steps, placing irregular vertices at first (usually processing a vector field), then trying to find an optimal connectivity, sometimes called *singularity graph*. When irregular vertices are not properly connected, it is likely to have separatrices that wind the shape multiple timess, even crossing themselves, before meeting their endpoint (Figure 4.3.b). Quad layouts with such bad connectivity will fail to meet any quality criteria, thus they are undesirable and need to be corrected using structure optimization approaches. Similarly to what has been said for irregular vertices, the number of patches is important and usually preferred to be small, or at least strictly dependent to the number of semantical components of the object being remeshed.

It is usually difficult to fulfill all these requirements at the same time, hence different approaches try to find a good balance between them.

Chapter 5

Related Works

In recent years there have been a number of different approaches to obtain quad layouts of good quality. They can be roughly categorized into :

- Global Parameterization methods
- Field Tracing methods
- Structural optimization methods
- User-assisted methods

Quad layout computation is a slightly, but substantially different problem than quad-remeshing. The purpose of quad remeshing is to produce a quad-dominant or pure quad mesh, directly manipulating the input surface mesh (usually triangulated) or taking as input an existing patch layout or graph singularity [TACSD06, RLL*06, BZK09].

Conversely, quad layout computation aims to produce a patch structure from which a high quality semi-regular quad mesh can be trivially computed. This state of the art discussion will focus essentially on quad layout computation hence only the methods which aim to its computation or whose results produce a structured quad mesh will be discussed, referring to [BLP*13] for an extensive review of quad mesh generation.

5.1 Global Parameterization methods

The main idea behind Global Parameterization techniques is to find a mapping between a canonical quadmesh, the 2D Cartesian grid, and the original surface (see Figure 5.1). This can be intuitively explained as the problem of finding a nice way to cut open the input surface and flatten it onto a gridded plane; once the correspondences between the grid and the flattened

surface are found, the grid can be seamlessy mapped back onto the surface fulfilling special compatibility conditions on surface's cuts.

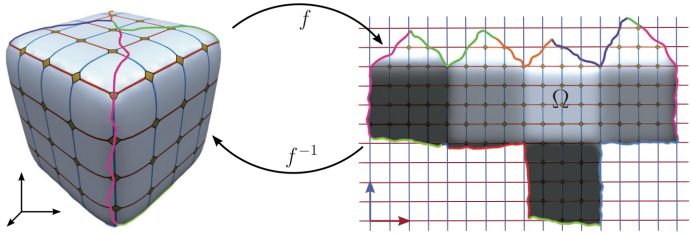


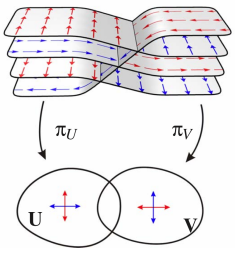
Figure 5.1: *Main idea of parametrization based methods. It consists in mapping the canonical quadmesh formed by the 2D Cartesian grid onto the input surface. Image courtesy of [BCE*13]*

Some global parameterization approaches take as input a coarse patch layout [TACSD06] , while others obtain these layouts optimizing the connectivity of the singularities of vector fields computed onto the surface.

The work of Ray and colleagues [RLL*06] is fundamental in this category since it presents a method to compute a global parameterization of a triangle mesh, using periodic potential functions aligned to two orthogonal vector fields, which are typically an estimation of the principal curvature direction of the surface. These periodic functions provide a smooth parameterization of almost the entire surface, except near the singularities of these functions. The patch boundaries are then constructed following the functions' isolines.

QuadCover [KNP07] extends this idea defining an automatic method for computing a global continuous parameterization starting from an input surface and a user-provided *cross-field*.

Cross-fields are *4-Rotationally symmetric fields* which describe at each point $\mathbf{p}$ of the mesh 4 replicas of a vector $\mathbf{u_p}$ rotated by an angle of $\pi/2$. The main innovation of this work lies in the manipulation of the cross-field that is simplified to a single vector-field on a branched covering of the input surface, see inset for a visual example (image courtesy of [KNP07]) . The branch points of the covering space lie at the branch points of the cross-field, thus leading to the singularities of the final quadrangulation.



Cross-fields are currently extensively used as a guidance for the generation of quad meshes and quad layouts since they exhibit the same types of singularities that can be observed in quad meshes. Due to their relevance recent works [ECBK14, PPTSH14] focus on generating good cross-fields.

Thanks to this concept it is possible to subdivide the quad mesh generation problem into simpler and more manageable problems.

In particular, Bommes [Bom12] identifies three subproblems :

- Generation of an orientation field
- Generation of a sizing field
- Generating a Quad mesh, accordingly to orientation and size prescribed by the previously computed fields .

An important work exploiting this pipeline is **MIQ** [BZK09] in which, however, a costant sizing field is used. The pipeline used by Bommes and colleagues requires only a small set of relevant directions, which can be automatically detected (e.g. tresholding dihedral angles) or manually defined. This direction constraints are then used to generate a smooth interpolating cross field, that is the first subproblem. In the second part of the pipeline, that is a global parameterization problem, the mesh is cut open and flattened having all the cross-field *singularities* (i.e. the points in which the vector or cross field is not defined) lying at its boundary. Two piecewise linear scalar fields u,v, with their gradient aligned to the cross-field, are then computed. This results in a parameterization that is compatible at the cuts and has its singularities mapped to integer positions along this boundaries. Both subproblems are formulated as mixed-integer problem, i.e. a linear problem where there are both continuous and discrete variables. Key contributions of this approach are both its mixed integer formulation and the use of a novel greedy solver for this kind of problems. This approach allows to obtain nice quadrangulations with both quads and separatrices with a good alignment w.r.t. the initial direction constraints, which are usually the salient shape's features.

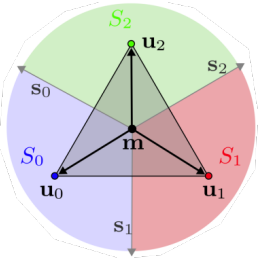
It may seem straightforward to produce coarse quad layouts using one of the discussed global parameterization methods with a sizing field which imposes a large target edge length. It also seems apparently easy to try to directly build a quad layout, given a set of singularities of known valence, by incrementally constructing the separatrices finding their trivial connections. However these two approaches will give rise, in the first case, to unreliable results due to parameterization degeneracies, and in the second case, to complex issues that need to be managed.

In the the first hypothesis, degeneracies like: flipped triangles in the domain or entire edges or triangles mapped to a single points typically occur when the desired edge length is larger than the distance between singularities. Incremental separatrix construction, instead, would give rise to a number of complex issues that must be managed, such as guaranteeing that all patches are quadrilaterals, which can difficultly be solved without introducing new singularities and complexity.

This two examples show the necessity of specialized quad layout computation algorithms, seen that they cannot be reliably obtained using previous global parameterization approaches.

In [BCE*13] Bommes and colleagues presented a novel global parameterization approach which guarantees to always produce pure quad meshes by construction using a convex Mixed-Integer Quadratic Programming formulation, thus allowing to produce coarse quad layouts specifying large target edge lengts.

The fundamental idea is similar to the one exposed by previous works, but in order to guarantee that a map is in the class of *Integer-Grid Maps* which correctly stitch the grid of integer iso-lines to a valid quad mesh, there are additional conditions that must be fulfilled. In particular, one of these conditions eliminates the possibility of having flipped triangles defining a trisector for each triangle and constraining its vertices to stay inside its assigned sector. See inset for a graphical explaination (image courtesy of [BCE*13]).



Moreover, in order to keep the computational times acceptable, the problem is solved for a carefully decimated version of the mesh. Supplementary conditions are added in order to separate singularities, avoiding that different singularities can be mapped onto the same integer location. The newly defined problem is then solved in order to produce the final quad layout.

5.2 Field tracing methods

Field tracing techniques, in contrast to global parameterization techniques, aim to produce quad layouts tracing their separatrices over the input surface. These separatrices, as previously stated, cannot be reliably constructed in a trivial way obtaining all quadrangular patch layouts.

Campen and colleagues [CBK12] addressed the problem of directly tracing a quad layout working in a dual setting. Their approach can be roughly summarized as follows: first a collection of transversally crossing loops on the surface is generated accordingly to a given cross-field, then the final quad layout is then obtained dualizing the loops' graph. In order to build these *dual loops* on the surface, a *four-sheeted* and a *two-sheeted branched coverings* are constructed following [KNP07]. Loops are then build tracing geodesic curves over the surface following the given cross-field, but allowing slight deviations from it, in order to optimize their path both for shortness and direction fidelity. Further constraints based on the 2 and 4 coverings are imposed in order to obtain a structurally valid loop's arrangement and loops are greedily chosen aiming to maximal singularity separation. The layout primalization is then obtained tracing field-guided geodesics which start from the primal nodes, which are dual faces of valence $\neq 4$ and are allowed to *run* only through the corridors provided by the dual layout. The

original input surface is then parameterized over the quad layout by firstly building a fine quad mesh, by refinement of the quad layout, then optimizing it by iterated parameterizations on the refined quad mesh.

More recent works belonging to this category are [MPZ14,RRP15], which directly trace polylines on the surface with the guidance of an input cross-field.

[MPZ14] trace polylines using motorcycle graphs [EGKT08] in order to produce an initial quad patch partition, this partition is then made globally consistent removing all the degenerate configurations that can harm the global parameterization that is computed as last step of the proposed approach.

[RRP15] uses a similar approach. Starting from the singularities of the input cross-field the separatrices are traced following the isolines of an initial parameterization. A graph $\mathcal{G}$ composed of the singularities and all the polyline crossings is then constructed and duplicated in order to find a *perfect matching*, that is a subgraph $\mathcal{Q}$, containing all the original nodes of $\mathcal{G}$, in which each edge vertex can belong to a single edge. Further constraints are imposed on the construction of $\mathcal{Q}$ to avoid non-quad solutions.

5.3 Structural optimization methods

A completely different approach to convert a triangle mesh to a structured quad mesh is *Structural Optimization* which relies essentially on a more or less trivial tri-to-quad mesh conversion (e.g. using one iteration of Catmull-Clark subdivision [CC78]) which typically produce a non-structured quad mesh, and a subsequent manipulation of the obtained mesh in order to simplify its global structure (i.e. the quad layout).

Two notable examples are [BLK11, TPC*10]; both rely on a small set of local operations aimed to simplify the mesh structure removing quad helices (see Figure 4.3). The application of these local operations is prescribed to keep an all-quadrilateral mesh.

[BLK11] identifies all the helices present in the quad mesh and removes them marching along face loops only using edge collapse operations, declined in different ways, which must obey to a finite-automaton which guarantees that, once the entire helix is traversed and removed applying these operators, the mesh is still composed of quadrilaterals only.

In [TPC*10] a different set of operators is proposed, essentially combinations of edge flips and edge collapses, which are driven by a measure called *homeometry* used in order to favour operations on regions of zero Gaussian curvature, which should usually contain regular vertices. All operations preserve the mesh topology and are interleaved with a tangent space smoothing. Tarini and colleagues proposed also a tri-to-quad mesh conversion method which halves the number of mesh faces and produces a pure quad mesh also

in case of odd number of faces.

5.4 User-assisted methods

State of the art methods for automatic quad remeshing and quad layout computation are able to reliably produce high-quality quad meshes, however they are all guided by purely geometrical information, such as principal curvature directions, which can derange the computation and produce overly complex layouts or fail to align the patches with semantical features of the mesh, especially when these features are not sharp.

Strangely, user assisted methods have been proposed only in very recent times. This can be due to the intrinsic complexity of editing quad layouts due to the potantially global effects of each local change. In 2013 Takayama and colleagues proposed a sketch based approach [TPHS13] for user-friendly definition and manipulation of quad meshes over triangulated surface. The proposed tool allows to quickly create polygonal patches that can be stitched together retaining a global consistency of the quadrangulation. The system allows to specify the number of subdivisions for each patch edge, as well as desired edge flows, generating the irregular vertices needed to accommodate such patch composition and preferred edge flow, using a novel approach contextually proposed [TPSH14]. This approach has been subsequently extended [MTP*15] in order to store the choices the user has made over time and propose her, for each patch defined over the surface, the solution the user would most likely choose between a set of possible configurations, enumerated using an approach similar to [PBJW14].

Another recent approach for user assisted quad layout generation is [CK14a] where the user is requred to iteratively add *Dual Strips* over the input surface until a satisfactory covering is obtained. These dual strips are inspired by their previous work [CBK12] and can be seen, in fact, as as set of parallel *dual loops*, giving a guess about the edge flow of the resulting layout. In order to build a satisfactory, for the system, covering of the surface the user has some visual indications of which regions require to be covered by a strip, essentially according to the current covering and curvature estimations. The system always proposes valid strips only and allows the user to refine them at design time. Differently from the other approaches of this category, once the layout is computed, the user has no direct control to refine or change the computed solution.

It is important to notice also that there are limitations in the quad layout manipulation in the other approaches too, users are in fact allowed to create patches, define edge flows, but they cannot freely manipulate the overall structure of the layout. This is a desired behaviour, due to the well known complexity of designing a good quad layout, since each local change can harm the quality of the global structure. It is then desirable to expose

to user manipulation only a proxy or a limited set of choices which can modify this global structure only in an ameliorative way.

5.5 Polycubes

Another interesting, while indirect, way to obtain a quad layout is computing a PolyCube [THCM04] of the input shape, that is a polyhedra, roughly resembling the original shape, with all its faces rectangular and axis aligned. Their were originally designed for texture mapping applications, however they can be used as base-complexes for parameterizing closed surfaces for different applications such as hexahedral meshing [GSZ11].

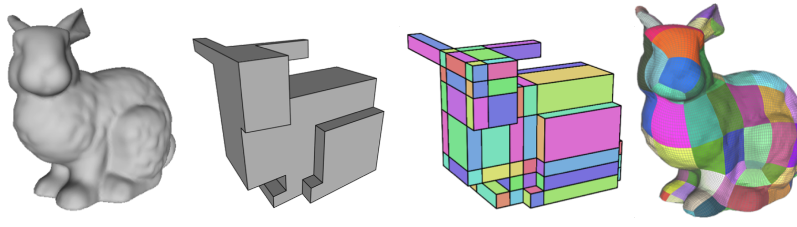


Figure 5.2: *The original model (a) and one possible polycube (b). Tracing all the separatrices over the polycube surface a pure quad layout can be obtained (c) and mapped back onto the original surface (d).*

In Figure 5.2 is depicted the typical pipeline, where from the original model a polycube map is computed, mapping the original surface to a very simple base complex. Since all the faces of the polycube are quadrilateral, the quad layout can be obtained tracing the separatrices between irregular vertices. The resulting layout can be mapped back onto the original surface. Although quad layouts obtained using polycubes have been will be presented for comparison purposes, being out of the scope of this thesis, they will not be discussed, referring to a recent work [LVS*13] for a state of the art review.

Chapter 6

Quad Layout computation guided by the curve-skeleton

Starting from the approach to polygonal modeling described in the introduction of this Part and from the need to take into account semantical reasonings in order to produce high-quality quad layouts, the idea behind the approach proposed in [Usa15] is to use the structural and semantical information provided by the curve-skeleton to guide the quad layout computation, mimicking a subdivision&extrusion process.

The pipeline (see Figure 6.1) can be summarized in 4 steps:

1. A triangle mesh and its curve-skeleton are taken in input.
2. A set of hexahedral boxes are placed in correspondence to branching nodes (**Bn**'s) and leaf nodes (**Ln**'s) of the skeleton. The optimal orientation of these boxes are then computed accordingly to the outgoing skeletal arcs.
3. These boxes are connected together with *tubes* extruding their faces along skeleton branches.
4. A parametrization of the input shape on such quad layout is then computed and provides a base mesh to model the object at hand. The parameterization is a two step procedure, composed by a naive projection and an optimization stage.

The quad layout produced with this method is ready for further refinement without the need for extensive retopology. Due to the use of the curve-skeleton as a guidance for the quad layout computation, the class of

shapes that can be addressed with this method is mainly composed of objects that can be roughly approximated by an assembly of *generalized cones*. Hence the method is particularly suitable for animation characters, but it can be used with good results for each object for which a curve-skeleton provides a precise definition of its structure.

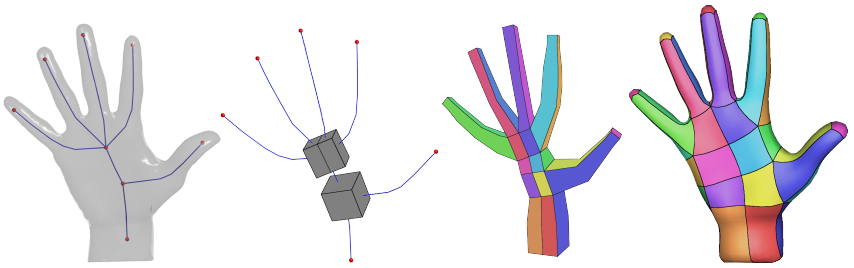


Figure 6.1: *a) The input is twofold: a triangle mesh and its curve skeleton. b) an hexahedral box is centered at each node of the curve-skeleton. c) boxes are connected following skeletal arcs in order to build a tubular structure. d) the input surface is parameterized over the tubular structure in order to obtain the final coarse quad layout.*

6.1 Contribution

The main contribution of this work is the derivation of a *pure quad layout* from a triangle mesh and its curve skeleton. Another contribution of this work is the definition of an intermediate structure that allows an automatic and manual manipulation, with a simple and intuitive metaphor, of the quad layout.

The presented method is automatic, fast and does not require any parameter setting. While its application is restricted to objects that can be described by a curve-skeleton, if a *good* curve-skeleton is provided the method can be applied to objects of any genus. This method can be applied in a fully automatic way, but also supports interactive techniques that allow the user to change the structure of the layout at interactive time, by improving the choices of the automatic system with purely local manipulations.

6.2 Method

The final quad layout produced by this method will be obtained starting from the aforementioned tubular structure (step 3 of the pipeline), aiming to

obtain a layout that is aligned with its tubes. The layout is expected to be fully regular along tubes, while singularities will appear only at branching elements and boxes placed at the terminal nodes of the skeleton (step 2 of the pipeline). In order to set the structure of the branching elements, some observations can be made from the common practice in interactive modeling of quad meshes:

- Most irregular vertices should have valence 3 or 5; vertices with valence 6 are useful too, especially along symmetry axes (as the collapse of two coincident vertices of valence 5), while higher valences are rarely used;
- Tubes with quadrangular section are the most natural choice, since they can be extruded by cut-opening quad faces of a, possibly subdivided, hexahedral box; such an extrusion introduces exactly four irregular vertices of valence 5;
- The basic modeling primitive is a hexahedral box which provides a mesh covering a genus zero volume with a minimal number of irregular vertices (eight, of valence 3) and it can be arbitrarily subdivided by *gridding* its faces without introducing new irregular vertices; the possibility of subdividing a box implies that an arbitrarily number of tubes can be extruded from its faces.

These observations lead to the definition of the pipeline presented in the introduction that will be discussed in the following sections.

INPUT

6.2.1 Input

The input of the pipeline is a triangle mesh along with its curve-skeleton. The curve-skeleton can be computed with standard methods (see Section 1.2) or it can be modeled interactively (see Chapter 3) and it must endow the overall structure of the object which will be modeled by the obtained coarse quad layout. The curve-skeleton turns out to be a powerful shape descriptor within this context since it is able to convey both structural and volumetric information about the shape; having a skeleton which provides also the diameters of the maximal spheres tangent to the surface or a complete mapping of the surface to its curve-skeleton, may allow for a more or less accurate reconstruction of the original surface. However this is not a stringent input requirement. During the experiments different skeletonization algorithms have been used, especially [LGS12, TAOZ12, DS06] as well as manually designed ones using Skeleton Lab (Part II) which has also been used, in some cases, to apply fixes to the automatically extracted ones.

6.2.2 Branching Elements

The hexahedral box placed at each **Bn** of the skeleton are the fundamental primitives of this approach and their orientation will influence the final layout. Explaining the concept of their work [CK14a], Campen and Kobbelt highlighted that one key question for a design process where a user is included in the loop is :

What should be the atomic operations provided to the user?

In their case the atomic operation is the placement of the so-called *Dual Strips* which provide a nice proxy for designing the structure of the quad layout giving a strong guess of which will be the resulting structure and edge flow.

In the case of the approach being presented here, the atomic operation is the orientation of one **Bn**'s box, which is a purely local operation but allows to change in a safe way, the overall layout composition and the final edge flow. The task of defining the orientation of these boxes can be entirely delegated to the user, however this approach can be plausible only in case of shapes for which the number of boexs is small; for complex models fine tuning these orientations can be tedious.

Typical, but not exhaustive, examples for three branches meeting together are T- and Y-branchings, and "tripods" (formed by branches that are nearly mutually orthogonal). Such configurations are easy to realize with cubic joints (see Figure 6.2). Note that in Y-branching the upper branches pierce the same face of the **Bn** box, thus requiring this face to be split into two facets; such a split shall propagate along the complex to avoid T-junctions.

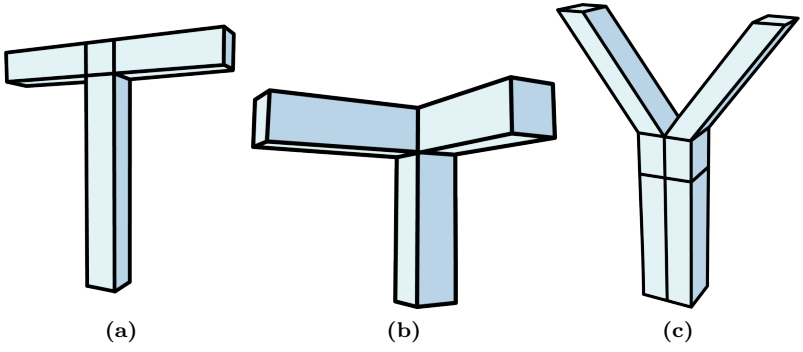


Figure 6.2: *a,b) T- and Y-branchings; c) tripod*

For branching nodes of higher degree, the combinatorics of configurations

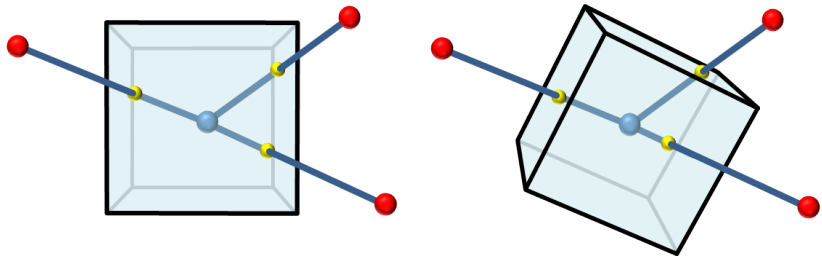


Figure 6.3: *An axis-aligned hexahedral box has all the three branches of the skeleton piercing the front face (left). Such a configuration would require splitting the front face in three facets. After reorienting it, the three branches pierce three different faces forming a "tripod" configuration, with no further splitting necessary (right).*

may become rather complex, so it is needed to face the problem of selecting the most suitable configuration in a general way.

Since this part of the pipeline is basically an *inverse skeletonization*(see Section 1.3), it is well known that, the most challenging problem is the definition of the geometry at the **Bn**'s. In this case the geometry is always a cube with fixed position and edge length, having only the freedom to decide its orientation in space, which will determine how its faces will be subdivided.

The first approximation that has been attempted was a statistical analysis of the angles formed by the arcs incident at each **Bn** in order to chose a branched polyhedra from a limited set of templates derived from the T's, Y's, and tripods seen in Figure 6.2, similarly to what Zhang and colleagues did in [ZBG*07b]. This solution, however, was not general enough since the goal is to orient the boxes at **Bn**'s trying to keep their faces as orthogonal as possible with respect to the directions of incoming branches (see Fig. 6.3).

In order to do this, the following optimization problem is resolved independently at each **Bn**. Given the branching node n_i and the set of k_i directions of its incident arcs $\{\mathbf{d}^i_1, \ldots, \mathbf{d}^i_{k_i}\}$, the aim is to find the orthonormal basis $U_iV_iW_i$ that minimizes the function:

$$f(U_i, V_i, W_i) = \sum_{j=1}^{k_i} \left|\mathbf{d}^i_j \cdot U_i\right| + \left|\mathbf{d}^i_j \cdot V_i\right| + \left|\mathbf{d}^i_j \cdot W_i\right|$$

The **Bn** will then be a box axis-aligned to basis $U_iV_iW_i$. Note that each term of the summation achieves its minima when one of the three coordinate axes is aligned with the branch, and its maxima when the branch stabs any corner of the box. The summation acts as a trade-off for the (mis-)alignment of the different branches to the basis.

As can be noticed f contains absolute values, hence is not C^1 continuous. To improve the stability of the solution, it has been used an approach similar to [HJS*14], minimizing the following function for $\varepsilon \to 0$ [EAVD79]:

$$f_\varepsilon(U_i, V_i, W_i) = \tag{6.1}$$
$$= \sum_{j=1}^{k_i} \sqrt{\left(\mathbf{d}_j^i \cdot U_i\right)^2 + \epsilon} + \sqrt{\left(\mathbf{d}_j^i \cdot V_i\right)^2 + \epsilon} + \sqrt{\left(\mathbf{d}_j^i \cdot W_i\right)^2 + \epsilon}$$

This minimization has been encoded as a non-linear constrained problem, by imposing that U_i, V_i and W_i are unit-length and mutually orthogonal. This minimization problem has been resolved using Ipopt [WB06]. The basis $U_iV_iW_i$ is initialized by computing the PCA of the set of directions $\{\mathbf{d}_1^i, \ldots, \mathbf{d}_{k_i}^i\}$. The first iteration resolves the problem with $\varepsilon = 0.5$, each solution is taken as warm start for the next iteration halving the value of ε; four iterations are always sufficient for the purpose.

The orientation of the **Ln** boxes is trivially computed in order to have the single incoming branch piercing orthogonally one of the faces of the box. This leaves entire freedom to rotate around the branch direction, and it will be used to create a tube with no torsion.

6.2.3 Initial box subdivision

After the minimization problem has been solved, all the boxes associated to each **Bn** are oriented. Let B_i be the box associated to a **Bn** n_i with k incident arcs. Each arc a_{i_k} incident at n_i is associated to the face of B_i pierced by the branch containing a_{i_k}. When more than one arc are associated to the same face of B_i, that face is split into *facets* following patterns allowing for tubular structures corresponding to different arcs to be extruded independently. In order to do this, all the intersection points between a face f and its incident branches of the skeleton are computed: if f has b_f incident branches, f is splitted into b_f parallel rectangular facets. In order to decide the direction in which f will be split, the intersection points are projectcted to the two cardinal directions parallel to the sides of f (in the $U_iV_iW_i$ basis), and the chosen direction X_i is the one on which such projection spans a larger interval. Each facet of f is then assigned to its corresponding incident branch. An example of split box is depicted in Figure 6.4. Note that the resulting tessellation of the cube can have T-vertices (i.e., the two half-edges of the same edge are not split in the same number of portions); this issue is fixed later for the whole quad layout with a global process (see Section 6.2.5).

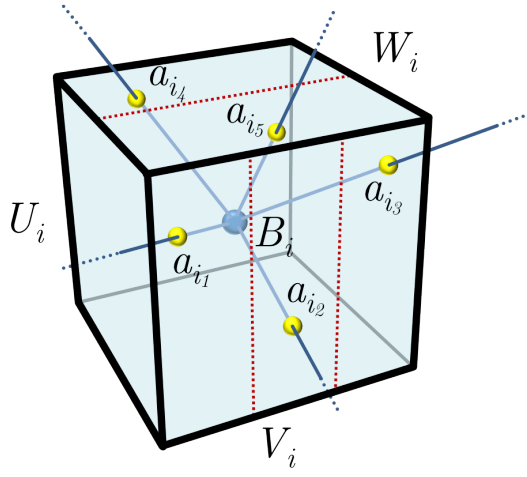


Figure 6.4: *A box with three branches (a_{i_1}, a_{i_2}, and a_{i_3}) piercing its front face, and two branches (a_{i_4} and a_{i_5}) piercing its top face. The front face is split into three facets along the V_i direction and the top face is split into two facets along the W_i direction. The resulting mesh has T-junctions.*

6.2.4 Extruding tubes

Once all the boxes are placed, oriented and subdivided accordingly to the skeletal structure, tubes can be extruded along the paths defined by the branches of the skeleton. All these tubes have quadrilateral section and connect **Bn**-**Bn** pairs and **Bn**-**Ln** pairs, aiming to minimize the torsion. Note that skeletons obtained using state-of-the-art methods are often composed by hundreds of nodes yielding an unnecessary complexity of the resulting tubes. It has shown to be beneficial for the performances and the final results a subsampling of the skeleton as proposed in Section 3.2.

To this aim, a square ring is associated to each **Jn** and such squares are then concatenated with longitudinal edges nearly parallel to the skeleton. The square ring associated to each **Jn** lies in a transversal plane, which is orthogonal to the tangent direction of the skeleton, estimated in a trivial way (see Section 3.3) while its rotation about the axis is set according to a Bishop frame [BWR*08].

Depending on the fact that the branch connects a pair **Bn**-**Bn** or a pair **Bn**-**Ln** the complexity of the operation changes. In case of a pair **Bn**-**Ln** the procedure starts at the **Bn** end of the branch, removing the facet pierced by the corresponding branch, and extruding the tube along it. A 2D reference frame is set in order to be aligned with the sides of the removed facet, the Bishop frame is set at each point along the branch by parallel transport of

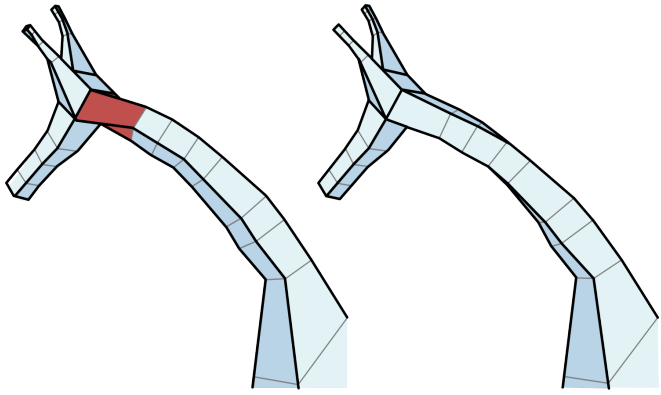


Figure 6.5: *If tubes are extruded along branches by using just the Bishop frame, all torsion occurs in a single red block (left image); we interpolate torsion along the neck, to minimize box-to-box torsion.*

the initial frame along the branch, traversing all the **Jn**'s to the **Ln** where an open box is centered. The tube is assembled by joining corresponding corners of the rings in chains, starting from the open face of the **Bn** to the open face of the **Ln**. The orientation of the 2D frames at all **Jn**'s along the path, as well as the **Ln**, set the orientation of the rings about the branch, so that no torsion occurs along it (see Figure 6.5, nose and ears).
In case of a pair **Bn**-**Bn**, the Bishop frame is set as in the previous case, by fixing the reference frame at one of the two ends. Conversely to the **Bn**-**Ln** case both the boxes have fixed orientation, hence there will usually be a mismatch between the orientation of the Bishop frame and the orientation of the box at the other end, for an angle θ on the frame's plane. Such mismatch induces a torsion on the tube, that is distributed by interpolating θ along the branch and rotating the rings at **Jn**'s accordingly. This can be done by arc-length parameterization along the branch (see Figure 6.5, neck).

6.2.5 Conforming tesselation

The tubular structure obtained at this point, is a mesh that is connected and watertight, but can have T-vertices depending on the subdivisions induced at each **Bn** box. It may happen in fact that some edges of the **Bn**'s may have been subdivided differently at the two sides (see Figure 6.6).

Previous approaches for T-vertices removal are able to generate a conforming mesh at the cost of two linear equations per face, imposing that opposite edges of the same face will be subdivided in the same intervals [BVK10, TACSD06]. However, these formulation are not general enough as they do not handle the case in which multiple faces are incident at both sides

of the same edge (see Figure 6.6), which in their case should not happen, but in this setting is very common. The following formulation is able to handle such case and requires a significantly smaller number of unknowns.

Both **Bn**'s ad **Ln**'s are cubes arbitrarily oriented in the space, therefore they naturally embed a 3D frame represented by the three orientations of the edges. For each node three unknowns are set, representing the number of splits required along each edge direction, in order to achieve conformity. Tubes are cylinders with a quadrilateral section, hence they can impose a subdivision only on two of the directions of the cube; follows that for each tube two unknowns are set, representing the number of splits along the two edge directions of its cross-section. The total number of unknowns required by the system is therefore $3n + 2m$, where $n = \#\mathbf{Bn} + \#\mathbf{Ln}$ and m the number of tubes. It is important to note that this number is *resolution-independent* and it directly depends only on the complexity of the *structure* described by the skeleton. Each tube, in fact, can be tessellated with an arbitrary number of quads along its length without affecting the size of the problem, while the complexity of the previous formulations is strictly dependent to the required resolution.

It is worth noticing that in this setting, also a formulation that is *resolution-dependent* would have been relatively compact due to the very coarse nature of the tubular structure. The proposed formulation is however more general and preferable in this setting.

Tubes are connected to boxes through a face or a facet (a portion of it). When a box and a tube meet at a face (facet) they must agree on the number of subdivisions of their edges along two directions. Using the example depicted in Figure 6.6 the tubes t^0, t^1, t^2 pierce the uv face of the cube C and t^0, t^1, t^2 must agree on the number of subdivisions along the directions u and v, generating four linear equations.

$$\begin{cases} C_u = t^0_u = t^1_u = t^2_u \\ C_v = t^0_v + t^1_v + t^2_v \end{cases} \tag{6.2}$$

The face vw of C is pierced by the tubes t^3, t^4. In a similar manner, to achieve conformity C and t^3, t^4 must agree on the number of subdivisions along the directions v and w, generating three more equations

$$\begin{cases} C_v = t^3_v + t^4_v \\ C_w = t^3_w = t^4_w \end{cases} \tag{6.3}$$

To produce a conforming mesh it is necessary to solve for the number of subdivisions of each cube and each tube, generating an homogeneous linear system $AX = 0$. Only non trivial solutions are acceptable, hence in order to find only solutions that prescribe a positive number of subdivisions for each element a further constraint is added stating that $X \geq \mathbf{1}$, where $\mathbf{1}$ is a vector with all entries at 1 and the inequality is intended component-wise,

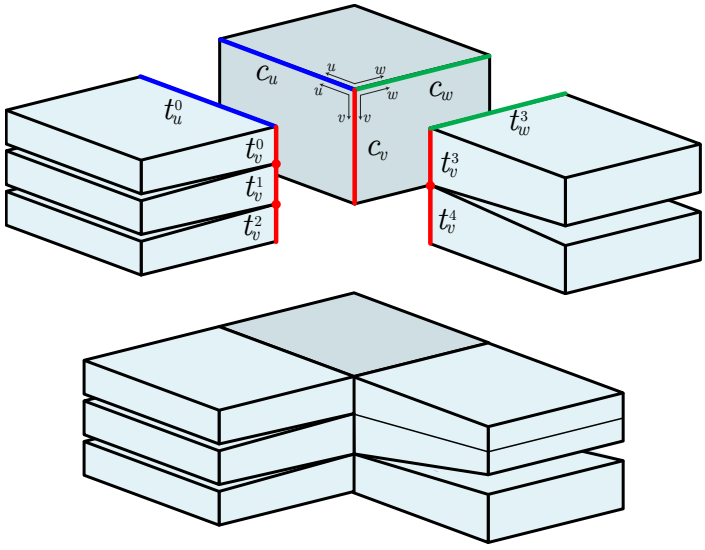


Figure 6.6: *We weld tubes on cubes in a conforming way (bottom) by imposing that the number of splits of a cube along each direction matches with the sum of the number of splits of each incident tube along the same direction.*

that is, each edge consists of at least one segment. The solution sought is the one that splits each edge into a minimal number of sub-edges while satisfying the constraints above. This gives the following ILP problem:

$$\begin{aligned} &\operatorname{argmin} \mathbf{1}^T X \\ &AX = 0 \\ &X \geq \mathbf{1} \end{aligned} \tag{6.4}$$

In the proposed implementation, the solution above is minimized by using [Gur13]. A peculiarity of this approach is that it is inherently *volumetric*, in the sense that the number of subdivisions prescribed generates surface meshes that can be trivially turned into full hexahedral meshes just by *gridding* the interior of each tube and each cube.

6.2.6 Barriers

The problem as previously described always admits a feasible solution for any genus zero object, that is, when the curve-skeleton of the object is a tree. However it may happen that objects of higher genus contain configurations that do not admit a solution, as the case depicted in Figure 6.7.

These situations, that in the context of this work have been called *Barriers*, arise because of loops of constraints that cannot be satisfied simultaneously. These barriers are located and removed one at a time by splitting the base complex at some point along the corresponding cycle, until the system becomes feasible. In the worst case, all cycles are split and the complex becomes of genus zero, thus guaranteeing to have a feasible solution. After a conforming complex is obtained, each split is rejoined by glueing corresponding ends at a common grid; the mismatch between abutting ends generates *lids* on the surface of the complex, introducing new irregular vertices (Figure 6.7).

The first step is detect cycles in the graph representation of the curve skeleton where just the **Bn** and **Ln** nodes are retained, and paths connecting them will determine the arcs. This is essentially a one-to-one match between arcs and tubes. This step is done at the cost of a depth-first traversal. Next, for each cycle c, all the equations containing the variables associated to its tubes are collected.

Let A_c be the sub-matrix of A composed by these equations only. To determine if c is a barrier it is sufficient to resolve the ILP problem restricted to constraints in A_c; c is a barrier iff this problem too does not admit a solution.

Barries can be removed by splitting the variables corresponding to cubes that appear along a cycle. Referring to the example in Figure 6.7, it contains a single loop consisting of a cube C and a tube t^1, corresponding to the equations shown in the figure, which also involve the tube t^2. The barrier can always be removed by splitting variables C_u and C_w into C'_u, C''_u and C'_w, C''_w, respectively, generating the new set of constraints as shown in the figure. Once the ILP problem has been resolved for the new variables, the cube is split according to the larger value between each pair of solutions C'_*, C''_*. The end of the tubes corresponding to the equation that involves the other variable, are attached to the cube in order to cover a sub-grid of facets of the subdivided cube. The other portion of the face of the cube is now on the surface, tagged *lid* in Figure 6.7.

The cube at which a given barrier is broken can be user-selected or chosen automatically with simple heuristics, i.e. selecting the cube with the least number of incident tubes. It is worth to note that the solution is not unique, in fact it is possible to match the split variables in different ways bringing to different configurations, all of them valid.

In the proposed implementation one of the solutions is taken in a systematic way, by setting a local reference system for each facet of the cube and selecting the sub-grid at its lower-left corner. More sophisticated criteria could be used to select the optimal solution among all the possibilities. While this approach allows to guarantee that a feasible solution can always be found, barriers rarely occur in practice. Most of the shapes in scope with this method, that are essentially characters, are of genus zero. Also higher

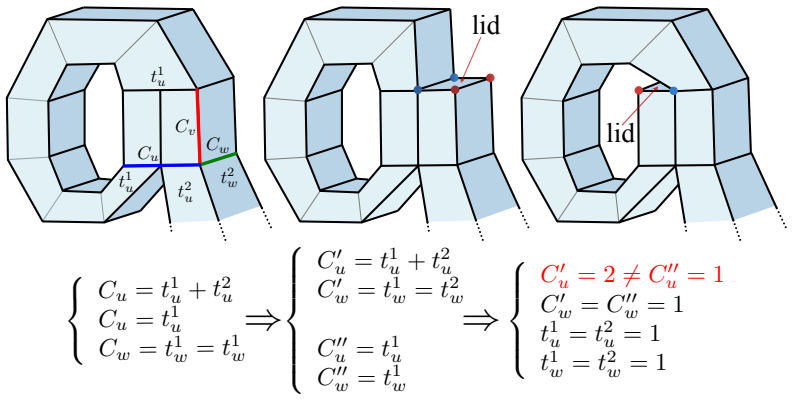


Figure 6.7: *The mesh on the left has barriers: constraints coming from the top and the bottom faces of the cube cannot be satisfied at the same time; the barrier is broken by splitting variables C_u and C_w corresponding to the blue and green axes; two possible solutions ca be found, as depicted on the center and right, which add a transversal facet (a lid) that introduces four more irregular vertices: two of valence 3 (in red) and two of valence 5 (in blue). At the bottom, the system of constraints that characterize the barrier is shown, the way it is modified by splitting variables and the resulting solution: the equations in red show the mismatch between values assigned to a split variable.*

genus objects against the proposed method has been tested (e.g. Fertility and Rockerarm) do not induce barriers.

6.2.7 Coarse Quad layout and mapping

The obtained conforming mesh can be further coarsened by the removal of all the edge loops that are transversal to the tubes. In other words, it is possible to remove all the edges that are not separatrices without affecting the final topology of the quad layout. The resulting quad-mesh $\mathcal{Q}$ conveys the topology of the final pure quad layout sought for the original mesh $\mathcal{Q}$. The last step of the pipeline requires the computation of a parameterization of $\mathcal{M}$ over $\mathcal{Q}$ in order to define a bijective mapping between the two.

To do this, to each vertex v of $\mathcal{M}$ is assigned a position on $\mathcal{Q}$ which is specified by the index i of a quad q_i of $\mathcal{Q}$ and a pair of parametric coordinates (u, v) normalized in $[0, 1] \times [0, 1]$ inside q_i. This per-vertex assignment implicitly partitions $\mathcal{M}$ into quadrilateral *regions*, each corresponding to a quad of $\mathcal{Q}$. Edges of $\mathcal{Q}$ are implicitly mapped into *arcs* traced over the surface $\mathcal{M}$. In a similar way, vertices of $\mathcal{Q}$ are implicitly mapped to general positions over $\mathcal{M}$, not necessarily to its vertices. The final mapping is obtained in two distinct phases:

1. Computation of a raw assignment by projecting $\mathcal{Q}$ onto $\mathcal{M}$.
2. Optimization of the mapping by interleaved and iterative refinement and resizing process.

Initial Mapping

At this initial step the assignment does not need to be accurate, it can produce fold-overs, distortions and suboptimal positioning of region boundaries on $\mathcal{M}$.

Skeleton fattening An embedding of $\mathcal{Q}$ is obtained placing the boxes, orienting them according to the skeletal arcs incident to the skeletal node to which each box is assigned, sizing them according to the radii of the their maximal spheres; in case the radii of the maximal spheres is not provided, their radii are roughly estimated using an approach similar to the one described in Section 3.3; this procedure creates a sort of polymerized *fattened skeleton* around the curve-skeleton.

Initial Projection Each vertex of $\mathcal{Q}$ is projected to $\mathcal{M}$ along the surface normal estimated on the fattened skeleton averaging neighbouring faces; if projection fails, the closest point on $\mathcal{M}$ is selected. It is however unlikely that the latter solution needs to be used, since the vertices are projected along their normal if they are inside $\mathcal{M}$ (as it usually is) or along their anti-normal if the are outside of it.

Subdivision Each face q_i of $\mathcal{Q}$ is iteratively subdivided using the Catmull-Clark subdivision scheme, obtaining a corresponding gridded sub-domain q_i. After each subdivision, newly created vertices are projected to $\mathcal{M}$ as in the previous step.

Back-projection The above iterative projection and subdivision approach is iterated a limited number of times, usually three. As last step $\mathcal{M}$ is projected over each face q_i and, thus, over $\mathcal{Q}$ using the same strategy of the initial projection.

Finally the edge loops along the tubular structure can be removed. At this point the geometry of $\mathcal{Q}$ is no longer relevant: in any further processing, $\mathcal{Q}$ is considered only as a collection of 2D rectangles provided with adjacency information; $\mathcal{Q}$ is *abstract* as defined in [PTC10].

Optimization

In this phase the parameterization is optimized both in the interior of each region and the locations and shape of the arcs, and the location of vertices

of the layout in $\mathcal{M}$. This optimization phase follows the technique proposed in the second part of [TPP*11]. This approach will be described for completeness, however other optimization strategies can be applied as well, for example the one proposed in [CK14b].

The procedure consists of an iterative relaxation of the positions in $\mathcal{Q}$ assigned to vertices of $\mathcal{M}$. At each iteration, $\mathcal{Q}$ is clustered in a number of *macro-region*, each consisting of a small group of adjacent quads of $\mathcal{Q}$. All the grouped quads have disk-like topology, Figure 6.8 shows an example of these clusters. A macro-region is categorized depending on the element from which the region grouping has been originated: a *Vertex* macro-region is composed of the 1-ring of a vertex of $\mathcal{Q}$ an *Edge* the macro-region is composed of two quads of $\mathcal{Q}$ sharing an edge, while a *Face* macro-region is a single quad of $\mathcal{Q}$. Parameterization inside each macro-region is optimized by means of Mean Value parameterization [Flo03], while keeping boundary vertices fixed. During each single iteration of this optimization, each triangle not entirely mapped into a single macro-region will be freezed, preserving the location of its vertices. At the end of the iteration, the macro-regions are disassmbled into the original quads of $\mathcal{Q}$.

At every iteration different assemblies of macro-regions are used, in order to guarantee that each vertex of $\mathcal{M}$ can undergo the optimization, i.e. appears in the interior of a macro-region. The partition is constructed, at each iteration, using a simple greedy algorithm: quads are incrementally assembled into macro-regions until no quad is left out; regions assembling starts from vertices of $\mathcal{Q}$ which has been frozen on the border for most iterations; remaining quads are assembled into *edge* macro-regions, using the same prioritization strategy; the remaining quads will form *face* macro-regions. The relaxation performed in the interior of the macro-region is also important in order to solve the possible mis-configurations that can happen during the initial projection, such as unfolding the fold-overs; in particular, this can be enforced in the optimization system by adding linear constraints, after [BCE*13].

Folds can be unfolded only if they fall entirely inside a macro-region. Although there no guarantees that every folded configuration will undergo the presented optimization in at least one partition, during the performed experiments there were no failure cases. In case of failures, ad-hoc countermeasures could be easily designed to force the assignment of an entire folded configuration, and its neighbouring vertices, to a single macro-region.

After [KLS03], in *vertex* macro-regions, quads are transformed by means of an exponential mapping, in order to flatten them on a plane. Although this mapping is perfectly conformal in the continuos case, it can occasionally increase the conformal energy, or even introduce new fold-overs, do to the discrete nature of the mesh; . This minor, and rare, technical difficulty could always be solved with a local and temporary refinement of $\mathcal{M}$ or more practical countermeasures such as rolling back of newly created folded

triangles or relaxation processes which result in a global increase of the conformal energy. The process is iterated until a the global conformal energy increase exceeds a predefined threshold, each iteration cannot decrease the total conformal energy, thus guaranteeing convergence.

Resizing An additional optimization, interleaved with the above, is used to determine the relative sizes of the abstract quads in $\mathcal{Q}$. This will be useful also to extract a semi-regular quad mesh (see Section 6.5.1).

As first step, the ideal aspect ratio r_i is found separately for each quad q_i, trying to minimize the total conformal energy of [LPRM02]. This sub-problem is solved in closed form for each interior triangle and area-averaged for each quad.

Next step is to find a globally consistent assignments of the widths and heights of all quads, which better satisfies the aspect ratios, up to a global scale. In the abstract domain, an adjacency defined between two sides of a pair of quads implies the equality of their lengths. Hence it is necessary to construct a system where each variable v_1, v_n corresponds to either the width or the height of a quad. The set of variables can be reduced with the above equalities. If a given quad q_i has v_w width and v_h height, the ideal result will be $v_w/v_h = r_i$, which can be rewrited as $\log v_w - \log v_h = \log r_i$. The corresponding overdetermined linear system is solved in the least squares sense (with one variable arbitrarily set to the unit) to recover the logs of all the variables, and thus the widths and heights of each quad.

6.3 User interaction

The whole pipeline described so far is fully automatic and it is fast enough to be executed at interactive frame rate on meshes of moderately large size. This allows to include the user in the loop allowing her to optionally

Figure 6.8: *Left-most: original mesh $\mathcal{M}$ color-coded according to the quads in $\mathcal{Q}$(left). Then: examples of different assemblages of macro-regions. Triangles of $\mathcal{M}$ which span multiple macro-regions (which are frozen during relaxation) are shown darkened.*

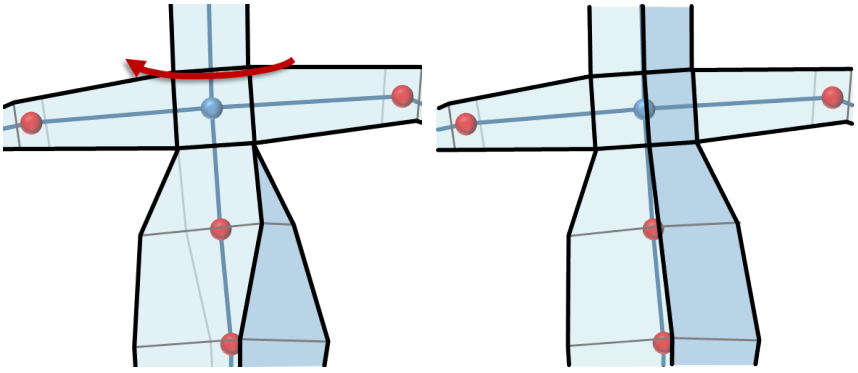


Figure 6.9: *The automatic cube orientation is generating an unnatural torsion (left). The user can pick the cube, select a rotation axis, and adjust it (right).*

steer the process controlling the choices taken by the system. As remarked throughout the thesis and stated by other authors, directly and freely manipulating a quad layout is a complex task even for an experienced professional. Therefore the forms of interaction allowed in this system have as target the intermediate structure used to generate the layout and not the layout itself. There are three ways to change the automatically computed structure.

6.3.1 Reorienting branching nodes

The user can select and change the orientation of a single **Bn** to change the layout of boxes, typically forcing skeletal arcs to pierce different faces inducing different box subdivisions, hence changing the structure of the coarse quad layout (see Fig. 6.9). In this way the output can be made to comply to additional desiderata, like for example the preservation of symmetry in the quad layout (see Fig. 6.10).

Adding branching nodes

By construction the original structure of the skeleton is used as is, hence are considered **Bn**'s only the nodes having at least three incident arcs. It can be useful, to account for sharp variations of the shape to insert a **Bn** box in correspondence of a node with only two incident branches. Recall that the skeleton captures the topology of the tubular parts without taking into account bending. Therefore, a branch is always modeled, by default, with a regular sequence of boxes, whose geometry is properly adjusted to follow the skeleton.

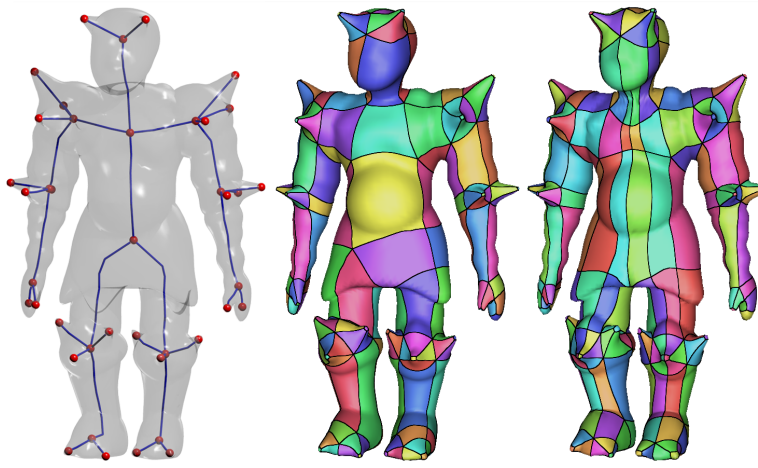
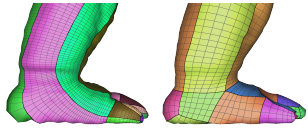


Figure 6.10: *Left: curve-skeleton for the Warrior dataset. Middle: the quad-layout produced with automatic box orientation. Right: an alternative quad-layout produced by manually reorienting some branching boxes, so as to emphasize the symmetry of the layout (Section 6.3.1).*



In some cases, is it however convenient to break the regularity by inserting a new **Bn**, as depicted in the inset where the heel is remeshed. Proably, using SkeletonLab such nodes would have been marked as articulations (**An**'s). This choice depends most often on semantics of the modeled object, and, therefore, the possibility to impose these additional **Bn**'s is up to the user. For instance, the trunk of an elephant is probably always best modeled with a straight sequence of boxes, no matter how sharply it bends, while an ankle of a human always requires an L-shaped joint, even if the foot is extended.

Retouching parameterization

While the other two forms of interaction are based on the intermediate structure, the third one allows the user to retouch the mapping during its optimization (Subsection 6.2.7), by constraining a vertex v of $\mathcal{Q}$ to be mapped over a specific position p on $\mathcal{M}$, assuming that p is currently mapped on a quad of $\mathcal{Q}$ which is adjacent to v; p is expressed as a triangle index and barycentric coordinates inside it.

To fulfill the constraint, the image of p is foreced to be mapped to the center of the macro-region built around v, in any subsequent optimization iteration that is performed on that macro-region. This translates to a simple

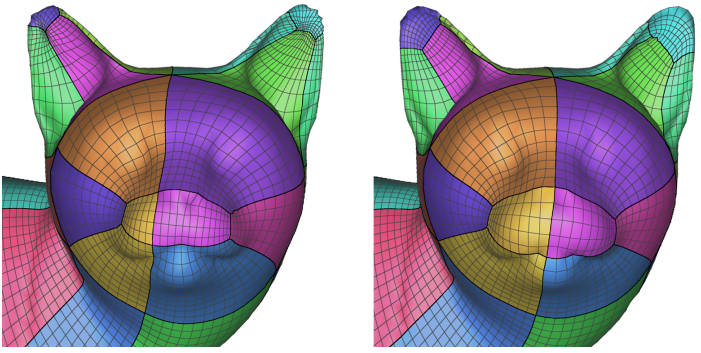


Figure 6.11: *Example of how a quad layout obtained automatically (left) can be improved with simple user intervention on the parameterization (right)*

linear constraint which can be included in the optimization.

It is to note that, in all other iterations, the triangle containing v is already fixed on the boundary. This interaction can be useful for both pin-pointing the exact location of irregular vertices of a remeshing (Section 6.5.1), or to get some control over the locations of seams of a UV-mapping (Section 6.5.2).

6.4 Results

The presented method has been tested on a set of about twenty different models, including a sequence of poses of the same shape. An exhaustive list of the results is summarized in Table 6.1 and the resulting layouts can be seen in Figures 6.10, 6.14, 6.12.

For all the models listed in Table 6.1 the method was able to produce a coarse quad layout during interactive sessions. These sessions lasted from less than one minute for the simplest model (i.e. the Cactus), to few minutes for the most complex ones (e.g. Dragon), where user intervention was needed to obtain optimal results. In all cases the automatic operations and the response to user intervention on the structure are almost immediate, up to the creation of the fattened skeleton and the quad layout topology. The parameterization part can take a few seconds, depending on the size of the input mesh. However it is needed to perform this operation just once, after the user is satisfied with the topology of the quad layout. Parameterization retouching is almost immediate as well and compatible with the user interaction. For the horse sequence, composed of 8 poses of the same mesh (same geometry), the presented method was able to produce the same exact layout for all the poses.

The coarse quad layouts obtained with this method, as well as the quad

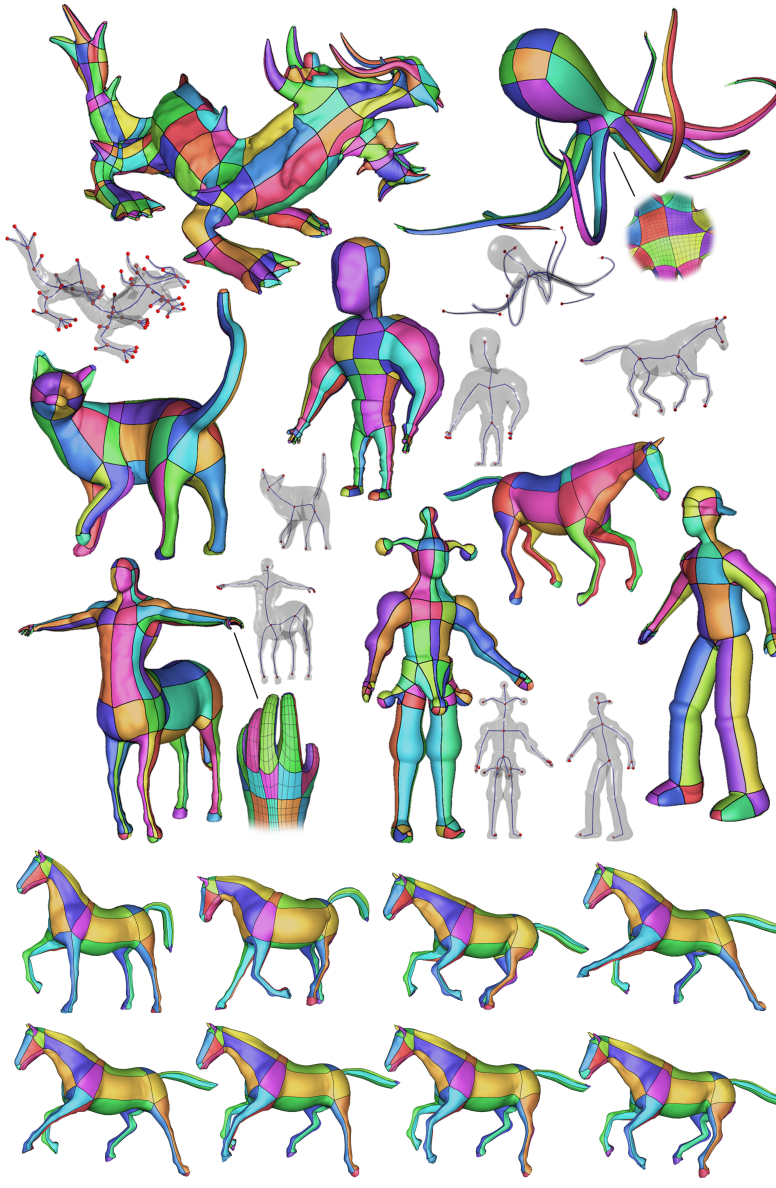


Figure 6.12: *A gallery of coarse quad layouts computed with the proposed method method. For each model we also show the curve-skeleton used as input (see also additional examples in figures 6.1, 6.14, 6.13, 6.15, 6.10 and 6.16).*

meshes that can be obtained from (Section 6.5.1), have most irregular vertices with low valence, 3 and 5, rarely with valence 6 or higher. Few objects of genus higher than zero has been tested as well and there never were the need to break *barriers*.

The presented method does not require any parameter from the user and the interaction is purely based on the appearance of the base complex, hence it can be run in a fully automatic way. Figure 6.10 show a comparison between a layout obtained in a completely automatic way and the same layout edited by the user in order to highlight the symmetries.

6.5 Applications

The quad layouts obtained with this approach can be directly used in a number of applications, here will be presented two important and common scenarios which are semi-regular quad remeshing and UV-mapping.

6.5.1 Semi-regular quad remeshing

It is possible to generate a quad remeshing of the original mesh, having the same set of irregular vertices of the corresponding quad layout, by performing a regular sampling over the parametric domain $\mathcal{Q}$. The resolution is arbitrary and it is determined by a user-defined scaling factor s: the widths and heights of $\mathcal{Q}$, as computed in Section 6.2.7, are scaled by s and then rounded to the nearest integer value. The vertices of the remeshing mesh are placed at integer locations of the parameterization. This is simply done traversing each face of $\mathcal{M}$ once, and producing all the vertices inside it with a rasterization approach. Such vertices are connected with a natural grid connectivity.

By construction, produced remeshings are pure-quad, conforming, and semi-regular (in the sense of [BLP*13]). See Fig. 6.13 for an example.

6.5.2 UV-mapping

The coarse quad layout directly provides a parameterization domain for UV-mapping, to be used for texture mapping applications both for the original triangle mesh $\mathcal{M}$. and a quad remeshing.

The rectangles of $\mathcal{Q}$ must be packed inside the global texture rectangle, by means of 2D translations of an integer number of texels, plus 2D rotations by a multiple of $\pi/4$.

Due to the very coarse nature of the produced layouts different packing strategies can be applied. One can use a simple rectangle-packing heuristic, as at the top left of Figure 6.15, or attach a few rectangles side by side, when the corresponding quads in $\mathcal{Q}$ are adjacent, in order to reduce the number of texture seams, as in the bottom left and right part of Figure 6.15. These

Table 6.1: *Summary of experiments. For each model the following information is showed: the number of faces in the triangle mesh (rounded to thousands); the number and type of skeleton's nodes; the number of domains and irregular vertices (valence 3, valence 5, valence 6 or more); the interactive operations of the user for reorienting and adding* **Bn***'s, if any. The name of each model comes together with a reference to the figure showing it.*

Model	kTri's	Skel. nodes (**Bn** , **Ln** , **Jn**)	Domains	Valence (3, 5, 6+)	User inter. Reor.	Add
Armadillo[6.14]	11	(7,18,50)	216	(72,48,8)	3	2
Big Buddy[6.12]	27	(4,11,66)	132	(44,24,6)	–	2
Cactus[6.15]	10	(2,4,22)	30	(16,8,0)	–	–
Cat[6.12]	56	(3,7,29)	64	(30,10,6)	2	–
Centaur[6.12]	31	(5,16,72)	166	(64,32,12)	1	–
Dinopet[6.15]	9	(6,12,79)	136	(48,24,8)	3	3
Dinosaur[??]	4	(2,6,191)	38	(24,8,4)	–	–
Dragon[6.12]	100	(25,47,121)	604	(188,132,24)	8	–
Elk[??]	48	(5,7,33)	72	(28,20,4)	–	–
Fertility[??]	37	(4,0,45)	46	(2,26,0)	–	1
Guy[6.12]	27	(4,5,44)	90	(34,22,2)	1	2
Hand[??]	21	(2,6,15)	48	(24,12,2)	–	–
Horse[6.12]	17	(3,8,36)	70	(32,12,6)	1	–
Joker[6.12]	27	(7,13,83)	154	(56,40,4)	1	2
Octopus[6.12]	33	(1,9,112)	56	(36,4,12)	–	–
Rockerarm[??]	20	(3,3,27)	28	(12,12,0)	1	–
Vessel[??]	99	(14,19,141)	304	(81,63,6)	1	–
Warrior[6.10]	27	(13,28,82)	234	(112,48,28)	4	2

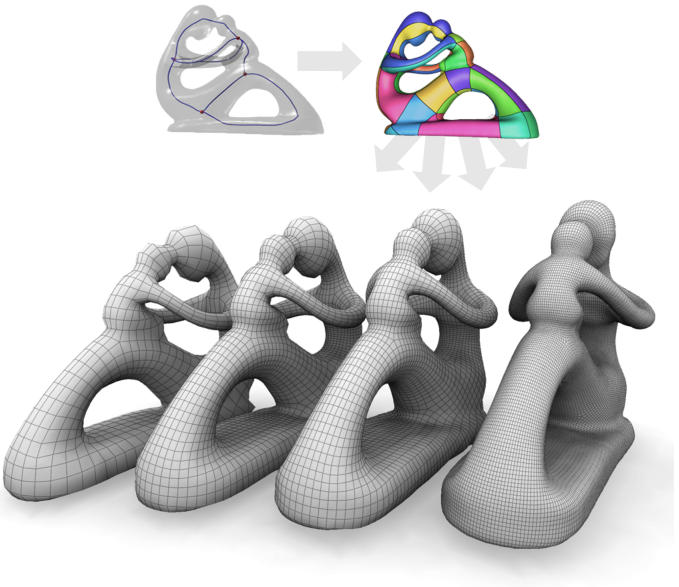


Figure 6.13: *Examples of semi-regular quad meshes at arbitrary resolutions that can be trivially extracted from the produced coarse quad layout (top-right); the curve skeleton which served as input (top left).*

tasks, which have a direct equivalent in typical UV-mapping construction, either by artists or by automatic approaches.

6.6 Comparisons

In this section will be discussed how the results obtained using this approach can be compared with state-of-the-art methods both for its core application, that is quad layout extraction and one of its applications, semi-regular quad remeshing.

6.6.1 Quad Layout comparisons

Extensive comparisons of the resulting layouts with the ones produced by other methods has been made, using the number of domains and the number of singularities as a quality metric. Table 6.2 shows comparisons with the two of the most recurring models in quad layout and quad remeshing computation literature, which are however not exactly in the scope of the presented method. For a visual comparison refer to Figure 6.16. Comparisons with polycube computation methods have been made, due to the possibility of

Model	[TPP*11]	[BCE*13]	[LVS*13]	[HJS*14]	ours
Fertility					
Domains	253	85	429	288	46
Valence 3	12	12	35	36	2
Valence 5+	36	36	59	†59	26
Rockerarm					
Domains	98	66	253	426	26
Valence 3	18	15	31	32	12
Valence 5+	18	15	31	32	12

Table 6.2: *Comparison on quad layouts obtained on the reference models most used in literature; the rows of the table report the number of domains of the quad layout, and the number of irregular vertices with their valence (†one valence 6 vertex).*

using polycubes to compute pure quad layouts for input meshes. It is worth noticing that the methods in [LVS*13] and [HJS*14] could have the ability to reduce the domains via tuning parameters. Note that, even if this method is designed as a support for animation, and thus mostly suitable for humanoid or animal models, it outperforms the state-of-the-art even on a mechanical model like Rockerarm and a model with high genus like Fertility.

A separate comparison with quad layouts extracted from the PAM models of [BAS14] can be seen in Figure 6.14. Such models are just quad-dominant and have most irregular vertices of valence 6 or 8 at branching structures and vertices of high valence at poles. For the sake of comparison, the quad layout of a PAM has been computed as follows: each polar cap is considered as a single domain; then domains from the remaining pure quad mesh are extracted by tracing separatrices incident at irregular vertices. The PAM layout contains 191 domains, 18 of which are polar patches; it contains 22 non-polar irregular vertices, all of which have valence at least 6, and 18 poles with valences between 4 and 36. Our layout contains 216 domains, 72 vertices of valence 3, 48 vertices of valence 5 and 8 vertices of valence 6. In spite of a slightly higher number of domains and singularities, the method here proposed produces a much more uniform layout and contains no vertex with high valence. Transforming a PAM into a pure quad mesh may require one step of subdivision, which would introduce further singularities of valence 3 and fragment the quad layout into many more domains.

6.6.2 Remeshing comparisons

In Figure 6.16 the results are compared with some state-of-the-art methods designed to produce a good quad remeshing, reporting also histograms

showing closeness of angles in the resulting quad mesh to a right angle. Further numerical results on the Rockerarm and Fertility models are reported in the table at the upper right of Figure 6.16. Note that while all methods tend to produce faces with nearly right angles on average, this method usually achieves a smaller relative standard deviation (RSD) in the distribution. In the Dinosaur dataset (middle of Figure 6.16), although the histogram bell of [ZJY15] is slightly narrower, their mesh has a number of quite sharp angles, resulting in a higher RSD (15.83% against our 7.78%). Only in the Vessel dataset shown in the bottom part of Figure 6.16, the method of [ZJY15] achieves a slightly better distribution of angles, but the layout produced with the proposed method contains about half a number of domains and they appear to be much better distributed on the surface of the object.

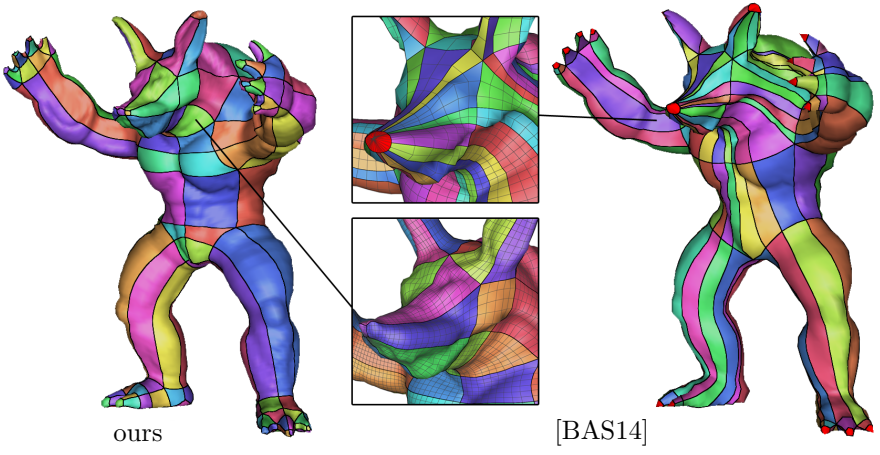
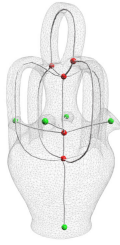


Figure 6.14: *Comparison between a quad layout obtained with our method (left), and a layout extracted from a PAM model (right). Polar patches of the PAM layout are depicted in red.*

6.7 Limitations

The presented method is intended for shapes defined as assembly of generalized cones, a class of objects suitable for models used in animation. Generally speaking, all shapes that cannot be described well with a curve skeleton are out of the scope of application of this method, for the same reasons, shapes with cavities (like a cup) cannot be addressed. Another example is the Botijo model (see inset), for which this method was unable to perform the initial mapping step due to the intrinsic characteristics of the shape. However, as shown in Section 6.4, reasonable results can be obtained even on some objects not directly intended for animation.



Due to the shape of classes addressed by this method, the method does not preserve sharp creases, even if some models would have had benefits from it. For example, this could have been added by snapping creases to borders of domains during parametrization, as proposed in [TPP*11]. Multi-scale detection would be necessary to capture just creases that are relevant at scales coarse as the desired quad layout, while disregarding fine details.

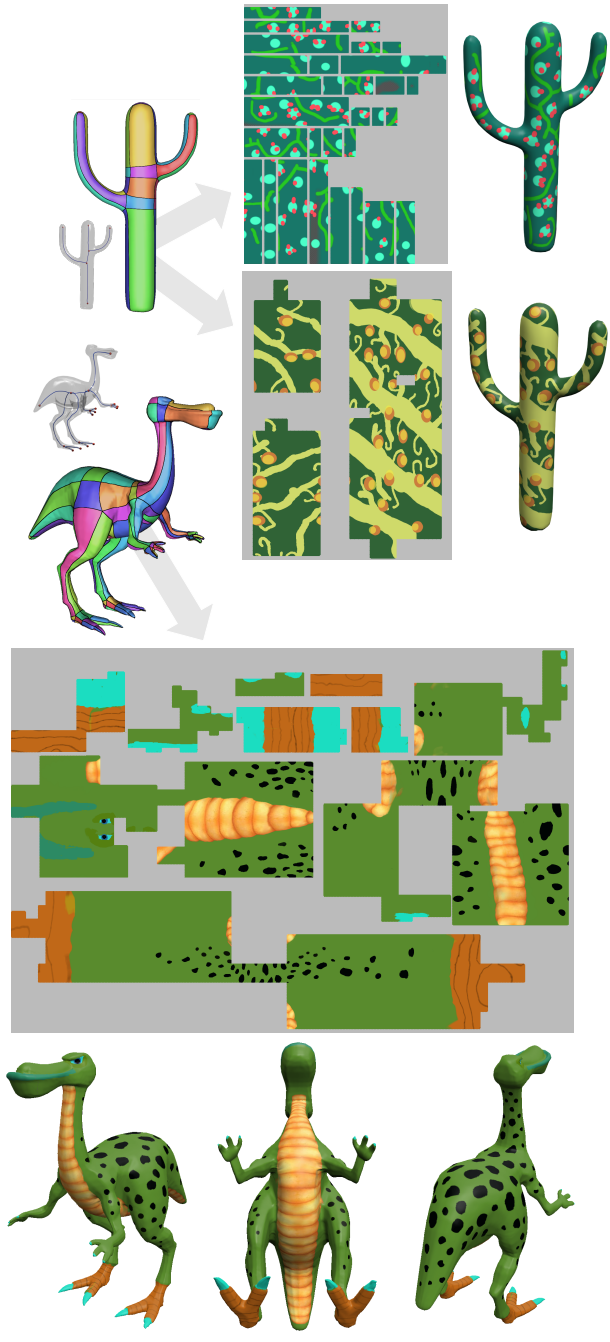
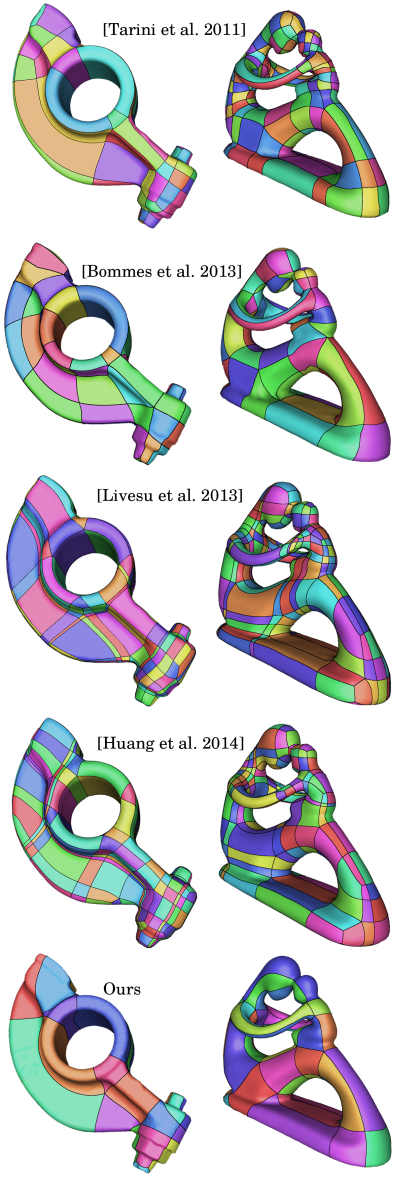


Figure 6.15: *Examples of UV-mapping obtained from the resulting coarse domains, with different texture packing strategies. The demonstrative textures were manually painted by artists using standard software.*



Model	Avg	RSD
Fertility		
[BZK09]	89.86	9.29 %
[TPP*11]	89.92	12.21 %
[BCE*13]	89.91	18.22 %
[LVS*13]	89.98	19.22 %
[HJS*14]	89.95	10.91 %
Ours	89.97	6.96 %
Rocker arm		
[BZK09]	89.95	8.79 %
[TPP*11]	89.97	7.50 %
[BCE*13]	89.92	18.57 %
[EBCK13]	89.74	9.21 %
[LVS*13]	89.98	16.24 %
[HJS*14]	89.96	10.82 %
Ours	89.96	8.45 %

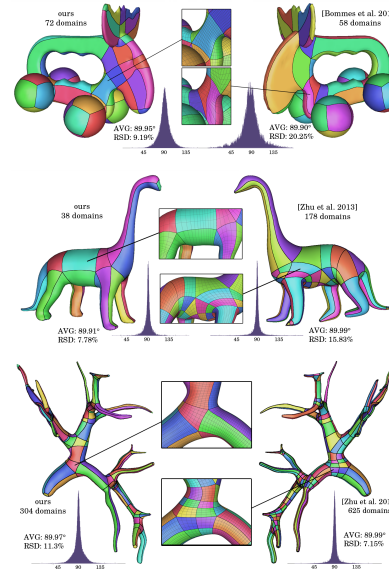


Figure 6.16: *Left: visual comparison of the layouts presented in the table at top right. Bottom right : Visual and numerical remeshing comparisons with [BCE* 13] and [ZJY15]. Overall, the method is capable of producing coarser layouts that lead to better semi-regular quad-meshes.*

Part IV

Procedural mesh composition using Polar-Annular-Meshes

The property of exploiting a well defined underlying structure is not exclusive of semi-regular quad meshes, also *Quad-Dominant Meshes* can be structured, in fact **Polar Annular Meshes** (PAM) are an example of that possibility. Briefly recalling the previously given description (Section 2.3), a PAM is a polygonal mesh consisting only of triangles and quads such that:

- each quad belongs to a single quad ring called *Annular Region*
- each triangle belongs to a single fan of triangles, centered at a vertex called *Pole*, denoted as *Polar Region* or *Polar Cap*.
- the set of faces incident to a vertex is always homeomorphic to a disk
- no t-joints are allowed

This representation is so restrictive that allows to naturally encode the shape's curve-skeleton in the mesh' geometry, it is in fact, always possible to trivially collapse the mesh revealing its skeletal structure. Following the mesh definition, edges can be subdivided into two precise categories: *spine* edges which are aligned with the local skeletal direction, and *rib* edges which are orthogonal to this direction and always form a loop. Rib edge loops can be always collapsed to their corresponding skeletal node; sequences of spine edges always connect two poles.

The structure of PAMs is particularly suited for modeling purposes, it is, in fact, easy to create new skeletally supported features of a given shape, or cut and paste parts of the shape without creating holes, being able to operate at the same time on both the geometry and the structure.

This part of this thesis will delve into the possibility of using PAMs for procedural modeling purposes, especially iteratively composing a shape starting from a set of user-defined pieces whose geometry defines both their shape and the way in which different pieces can be connected together. This work takes inspiration from different previous modeling approaches, especially grammar based methods for procedural modeling and part based modeling. The proposed approach tries to combine ideas taken from this two topics building a procedural composition system whose definition of rules should be easier for professional modelers than defining the rules of a formal grammar.

The work described by following chapters is currently a work in progress, hence preliminary results will be showed and discussed.

Chapter 7

Related Works

The work being presented is inspired by different previous works, although a complete review of the literature would be too extensive, hence a brief overview of the inspiring approaches will be presented instead. The fundamental work is obviously [BAS14] where the Polar Annular Mesh structure is introduced, see Section 10 for a deeper discussion.

The two main modeling trends which inspired this work are grammar based methods and part (or assembly) based modeling.

Grammar based modeling The use of formal grammars for visualization and modeling purposes started many years ago. In very simple words, the underlying idea is to define dictionary of symbols and a set of rules for transforming a symbol or a sequence of them into another sequence; it defines a string rewriting mechanism. At the end of the rewriting process, after a prescribed number of iterations or once that particular conditions have been satisfied, each symbol in the final sequence is interpreted as a command. This is the mechanism behind L-systems which have been introduced by Lyndenmaier in [PL90] as a formalism to express and visualize the growth of plants. The sequences generated using L-systems are then interpreted with a LOGO style turtle.

During the years the L-system formalism has been improved and extended in order to express more complicated behaviours, and as a way to concisely define rules for procedural modeling of plants [MP96], forests [DHL*98], buildings [PM01], and street maps [PM01], [TLL*11].

Another concept that have been succesfully used for architectural design is represented by shape grammars [SGSG71], which operate directly on shapes and have shown to be a powerful tool. While derivations with shape grammars can be done in a computer assisted way, usually require the user intervention for deciding the rules to apply. Different attempts to simplify and improve the concept of shape grammars have been made.

Particularly successful results have been obtained with CGA shape grammars [MWH*06], and generative mesh modeling [Hav05].

Part based modeling A *part* based approach to modeling can be subdivided into two specific problems: first, a set of *parts* need to be created, then these parts are glued together in order to produce the resulting shape.

This kind of modeling approach have a number of interesting aspects such as the possibility of gathering the modeled parts from multiple sources (e.g. handcrafting them, internet repositories, procedural modeling, ...), rapidly creating models from which take inspiration or producing an high number of variations starting from a limited set of parts and a target class of shapes in mind.

A practical example of a part based approach is the popular tool MeshMixer [SS10] which allows a user to quickly compose a 3D model starting from a library of parts that can be modified and glued together with a simple drag&drop mechanism.

Another example is [CKGK11], where the a workflow is similar to the one proposed by MeshMixer, but a system is included for contextually filter the repository in order to propose to the user only the parts that are semantically and stylistically compatible with the one being assembled.

A more sophisticated instance of the same concept can be found in the videogame *Spore* [Gam08] which offers to the player a *Creature creator* tool which combines freeform modeling for tubular parts and part composition features with the aim of creating the inhabitants of the virtual world.

Moreover a number of automatic methods for procedurally producing inspirational shapes starting from a database of parts and grammar-like rules for connection [GLXJ14] or mashups and variations of existing shapes [ZCOM13, FKS*04].

Chapter 8

Procedural composition of PAMs

The aim of this work is to define a simple but expressive procedural system for composing structured meshes. The main idea is to let the user provide a set of shapes that are used as building blocks, but also encode, using geometrical features, the rules which govern the way in which the individual parts can be combined together. This approach takes from grammar-based methods the idea of producing shapes that exploit a self-similarity, while in contrast to them it should provide to 3D artists a more familiar metaphor, than manipulating L-systems or shape grammars. The users just need to model the building blocks, carefully defining the geometry of the joining parts.

When comparing this idea with the inspiring works this approach shows two key aspects:

- **Flexibility**. The proposed approach is flexible enough to be used for different aims, ranging from the exploration of all the possible combination of a given set of shapes (without setting combining rules), to the possibility of carefully choosing combination rules, allowing for a controllable and predictable behaviour. Flexibility is intended also in the sense of the range of shapes that can be obtained since it is suitable for producing organic shapes, but it can also be used to build more regular ones.

- **Structure**. Current compositional methods do not produce structured meshes. The use of PAMs, instead, as representation allow to guarantee that at each step of the composition the resulting mesh naturally embeds its structure.

Next sections will introduce the terminology used in the rest of the

chapter, will explain the current work and will show some preliminary result. Following chapter will discuss the work that has been done, the preliminary results, while its possible evolutions will be discussed in the concluding chapter.

8.1 Terminology

In order to start explaining how this approach has been structured, it is necessary to introduce a small set of terms that will be used throughout the chapter.

- **Module**: The term *module* will refer to a part, a building block, represented as a PAM. Each *Module* $\mathcal{M}_i$ will define not only its geometry and structure but it will also encode the connection rules. When a module $\mathcal{M}_i$ is connected to the shape being composed it becomes an instance m_i of $\mathcal{M}_i$; this allows to recover for each part of the shape, at each step of the composition, to which instance of which module they belonged.
- **Pole**: the tip of a polar region (polar cap) of a PAM. They are particularly important in this context since the geometry of the polar caps will be used to encode the composition rules. A pole p have to be considered as an oriented point composed of its position $\dot{p}$ and its orientation $\vec{p}$, that is the skeletal direction at p.
- **Toolbox**: a user-defined set of *Modules* along with some high-level information. Each Module can be seen as LEGO piece and a toolbox as a LEGO box. A toolbox can also include some meta-data for each module $\mathcal{M}_i$ indicating:
  - Probability of $\mathcal{M}_i$ to be next module connected to the shape being composed.
  - Maximum number of instances of $\mathcal{M}_i$ that can be used. It can be also unbounded.
  - *Self-connectability*: A *flag* indicating if $\mathcal{M}_i$ is allowed to connect to an instance m_j of itself.
- **Main Structure**: the shape being currently composed. It will be usually denoted with the letter $\mathcal{H}$. At the start of the composition process it can be a prescribed shape, represented as a PAM, or a module taken from the toolbox.
- **Match**: pair of poles (p_i, p_j) where p_i belongs to a module $\mathcal{M}_i$ and p_j belongs to the main structure $\mathcal{H}$. A set of matches will define how a module $\mathcal{M}_i$ will be connected to $\mathcal{H}$.

- **Valid Match Set**: a *Match Set* $\mathcal{S}$ is a set of *Matches* and it is said to be *valid* if it obeys to all the connection rules.

Moreover, throughout this chapter $P_{\mathcal{M}_i}$ and $P_{\mathcal{H}}$ will respectively refer to the set of poles of a module $\mathcal{M}_i$ and the set of poles of $\mathcal{H}$.

8.2 Pipeline

At a high level, the pipeline can be decomposed in three steps:

1. The user creates her Modules; the only stringent requirement is that modules are valid PAMs. An optional requirement is that the poles' geometry is carefully crafted in order to define the connection rules, see Section 8.4.

2. The user defines a Toolbox, optionally providing for each Module its probability, maximum number of instances and self-connectability.

3. The procedure takes as input a Toolbox $\mathcal{T}$ along with the contained Modules $\{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n\}$. Optionally a *starter* module can be provided as well. The system iteratively:

   (a) Picks a module $\mathcal{M}_i$ from $\mathcal{T}$, for which the maximum number of instances has not been reached, and according to the assigned probability (if defined).

   (b) Tries to connect $\mathcal{M}_i$ to $\mathcal{H}$ using a match set that is optimal with respect to $\mathcal{M}_i$ and $\mathcal{H}$. See Section 8.5 for the discussion about how to determine the *optimal match set*. At the first iteration, if a starter module have not been defined, an instance of $\mathcal{M}_i$ becomes the starter module.

   (c) If at least a valid match set has been found, $\mathcal{M}_i$ is connected to $\mathcal{H}$ using the chosen match set. See Section 8.3 for details on how a module is connected to the $\mathcal{H}$.

The iterative procedure stops at the occurrence of one of the following events:

- A user-defined number of iterations have been completed.
- $\mathcal{H}$ does not contain poles.
- For each module of the toolbox the maximum number of instances has been reached.
- For each module of the toolbox is not possible to find a valid match set.

Similarly to what can be done using grammar based approaches, where different sets of production rules can be interleaved, using the proposed method, once the procedure stops the user is allowed to change the toolbox and restart the procedure. The information about the connection of an instance of a module to the main structure are kept throughout the composition process, allowing to identify, starting from a pole $p_a \in poles(\mathcal{H})$ the instance m_i^k to which the pole belongs to.

8.3 Connecting a module

One of the main reasons why PAMs are suitable for this kind of procedural composition is the ease of connecting two PAMs, hence two modules, together.

Given two modules $\mathcal{M}_i$ and $\mathcal{M}_j$ and a match (p_a, p_b) s.t. $p_a \in P_{\mathcal{M}_i}$, $p_b \in P_{\mathcal{M}_j}$, connecting $\mathcal{M}_i$ and $\mathcal{M}_j$ by means of the match (p_a, p_b) requires to remove the triangle fans at p_a and p_b and connecting with a new face each pair of edges (e_a, e_b) belonging to the borders of $\mathcal{M}_i$ and $\mathcal{M}_j$ generated by the removal of the polar caps (Figure 8.1).

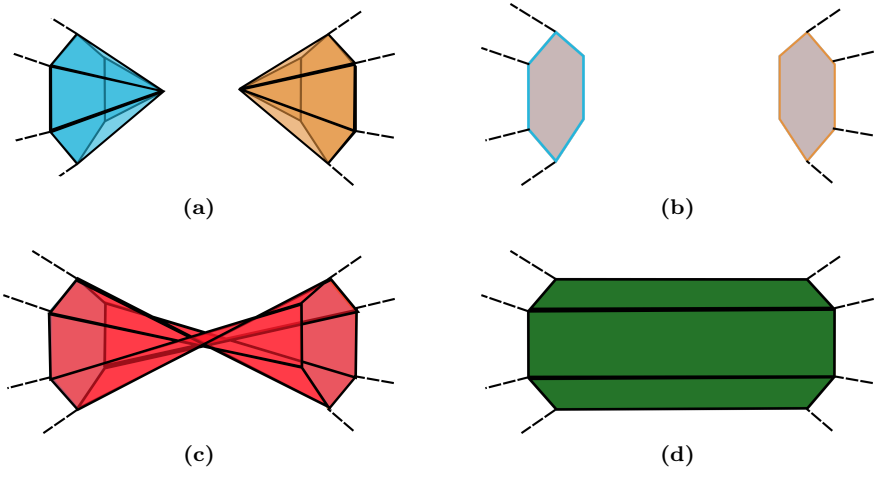


Figure 8.1: *a) Two aligned poles are going to be connected. b) their polar caps are removed. c) The two borders are connected naively causing a twisted geometry. d) Borders correctly connected using the simple heuristic presented.*

A naive connection of the two borders could cause that the resulting geometry is twisted around itself. A simple heuristic to prevent this issue is to keep an edge e_a of $\mathcal{M}_i$ fixed and walk the border of $\mathcal{M}_j$ in order to find a starting pair of edges such that, walking the two borders and trivially

pairing the edge on the two borders, the sum of their midpoint distances is the least. Once this starting edge pair if found, it is sufficient to walk the two borders connecting the other edge pairs with a quadrilateral face.

This operation guarantees that the obtained mesh is still a PAM. In the worst case all the poles of both the modules are connected together resulting in a pure quad mesh which, while still being a valid PAM, suffers from the same ambiguity of the torus example (see Section 10).

8.3.1 Orientation of the Match Set

Each module $\mathcal{M}_i$ is given in an arbitrary position and orientation, hence when connecting $\mathcal{M}_i$ to $\mathcal{H}$ is necessary to compute a transformation that moves $\mathcal{M}_i$ in such a way that is aligned w.r.t. $\mathcal{H}$ depending on the match set used for the connection.

In order to compute such transformation, a 6D version of [SH09] is used, where each 6D vector is an oriented point which encodes both the position and the skeletal direction of each pole. Solving the problem in a least square sense, a transformation that rigidly aligns the module $\mathcal{M}_i$ to $\mathcal{H}$ accordingly to a given match set $\mathcal{S}_i^k$ is obtained.

8.4 Connection Rules

One of the sources of inspiration for this work are grammar based approaches, for which a key aspect is the construction of the rules set. In the proposed approach the connection rules define if a module $\mathcal{M}_i$ can be connected to $\mathcal{H}$ by means of a match set $\mathcal{S}_i^k$. They can be categorized in different ways, depending on the scope at which they operate, and the fact that they can be user-defined, through poles' geometry or metadata, or imposed by the system. System-imposed rules are always active and every composition process is subject to them.

System-imposed rules

The general rules that superintend each composition process are simple and cannot be overwritten by the user-defined ones. They are :

- Two poles p_i and p_j can be connected together only if they have the same valence.
- A module $\mathcal{M}_i$ can be connected to $\mathcal{H}$ using a match set $\mathcal{S}_i^k$ only if the resulting geometry does not self-intersect. Section 8.6 explains how the composition process is tested agains self-intersections.

User-defined rules

User-defined connection rules can operate at two different scopes: single pole or entire module. Using the toolbox definition, the user can specify if a module $\mathcal{M}_i$ can connect to $\mathcal{H}$ through a match set $\mathcal{S}_i^k$ that contains pairs $(p_a,\ p_b)$ where $p_a \in P_{\mathcal{M}_i}$ and $p_b \in \mathcal{H}$ which belong to an instance of $\mathcal{M}_i$ that has been connected to $\mathcal{H}$.

However, the principal way in which a user can define the connection rules is through the pole's geometry. The first mechanism depends on the pole's valence; crafting a PAM, carefully choosing the valence of its poles, allows to restrict the poles and instances of modules to which it can connect to.

Moreover the shape of the polar cap, especially its cross section, allows to constrain the orientation of the two poles being connected, that is the rotation around the axis along two poles being connected. Figure 8.2 shows how the cross section of the polar cap can be used to encode orientation constraints. If the cross section is almost round (Figure 8.2.**a**) the orientation is not constrained, allowing two poles with same valence and round cross section to connect using any orientation. If the cross section is elongated on one or both sides (Figure 8.2.**c** and **b**) in order to connect, poles must be oriented in such a way that the elongated sides match; this allows two possible orientations (flipped by 180 degrees) in case **b** and just one orientation in case **c**. This orientation constraint is called *Elongation Direction*, it can be encoded as a unit vector orthogonal to the pole direction $\vec{p}$ and for a pole p it will be denoted as $\overset{\mapsto}{p}$.

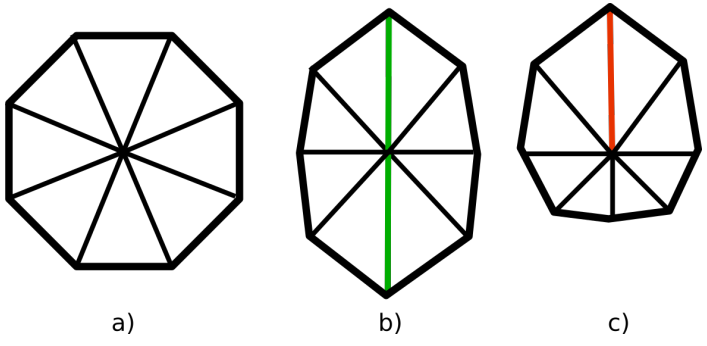


Figure 8.2: *Example of different annular region's cross section for a pole with valence 8. Each type should connect only to poles with the same cross section, aligning to their preferred orientation, if defined.*

This mechanism also works as a *compatibility* constraint since poles with cross section of type **a** can connect only to other poles with the same cross section type; the same holds for types **b** and **c**.

A problem that arises encoding rules with the pole's geometry is that a rule defined on a pole p_a of $\mathcal{M}_i$ will always allow the connection with the same pole on another instance of $\mathcal{M}_i$. In order to allow the user to prescribe that a pole should not connect with the same pole on another module's instance a simple formalism is used: this behaviour is allowed if the pole valence is even and disallowed if it is odd. Another solution would be to allow the use of vertex colors to be part of the definition of composition rules, e.g. self-connection is allowed for the white poles, but not for the black ones.

8.5 Optimal Match Set

The core of the proposed method is the problem of deciding, given a main structure $\mathcal{H}$ and a module $\mathcal{M}_i$, the optimal match set $\mathcal{S}_i^*$ to use in order to connect $\mathcal{M}_i$ to $\mathcal{H}$. In order to find $\mathcal{S}_i^*$, a set of trivial initial configurations (composed of a single match) is computed; then this configurations are expanded and the solution that optimizes the number of matches and a given quality metric is chosen.

Throughout the composition process, a kd-tree containing all the locations of the poles belonging to $P_{\mathcal{H}}$ is constantly kept updated in order to be able to perform distance queries.

First, a list of all the possible matches is composed, filtered from all the matches that do not satisfy the connection rules (i.e. different valence, different cross-section,etc) and randomly shuffled; then the space of all the possible valid configurations is explored.

Generating the initial configurations

For each remaining match (p_a, p_b) the transformation that aligns $\mathcal{M}_i$ to $\mathcal{H}$ with respect to the match is computed using the procedure defined in section 8.3.1. This transformation is then applied to $\mathcal{M}_i$. From a practical point of view this results in having $\dot{p_a}$ coincident to $\dot{p_b}$ and $\vec{p_a}$ and $\vec{p_b}$ that are parallel and with opposite direction; the direction $\vec{p_b}$ defines a rotation axis Ω which is used to compute a set of configurations:

- If the cross-section of p_a and p_b is almost round (Figure 8.2.a), n configurations are generated rotating $\mathcal{M}_i$ around Ω of an angle $(2\pi)/(n+r)$, where n can be defined by the user and r is a small randomly generated value, in order to allow for a larger variety of orientations.
- If the cross section is elongated on one side (Figure 8.2.a), only one configuration is generated after the application of a rotation around Ω that aligns the elongation directions $\overrightarrow{p_a}$ and $\overrightarrow{p_b}$;

- In case of a cross section elongated on two opposite sides (Figure 8.2.a), the previous procedure is applied, and two configurations, rotated of 180 degrees, are generated.

All the configurations that generate self intersections are discarded.

It is worth noting that each transformation is not required to be applied on the entire geometry of a module; it is sufficient to keep a reference of the skeleton of a module and apply the transformation to it. The **Ln**'s nodes (position and local skeletal direction) of a module's skeleton coincide with its poles. Additional information such as valence and cross-section shape and elongation direction must be stored along with the poles. The orientation of the cross-section is expressed as a unit vector orthogonal to the skeletal direction, and must be transformed along with the skeleton. In case of round cross section this vector will have zero-length, while a flag will indicate if the cross-section is elongated on both sides or not.

Building the candidate match sets

Starting from the list of the initial configurations the aim is to find, for each configuration, a maximal match set, that is a match set $\mathcal{S}_i^k$ containing, if possible, a match for each pole $p_a \in P_{\mathcal{M}_i}$.

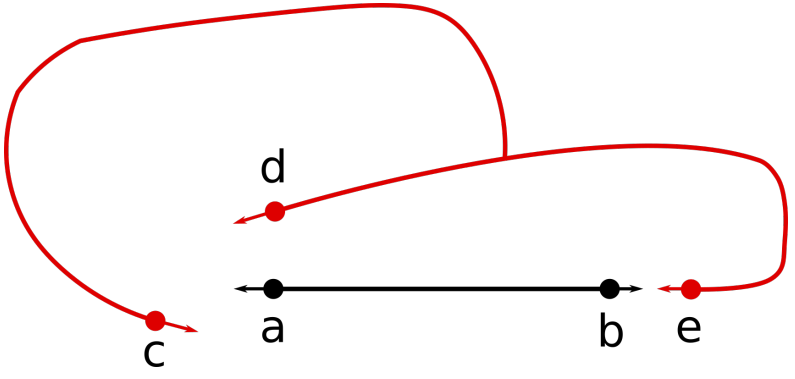


Figure 8.3: *Here a stylized version of the module (in black) and the main structure (in red) where only their skeleton and the pole's directions are shown. Poles* ***b*** *and* ***e*** *are used to compute the initial configuration. A match for the pole* ***a*** *is sought;* ***d*** *is not a good candidate since its poler direction is not opposite to* ***a****'s direction, hence* ***a*** *will be mapped to* ***c****.*

The match set $\mathcal{S}_i^k$ initially contains only (p_a, p_b); the other matches are added finding, for each pole $p_\alpha \in P_{\mathcal{M}_i}$, its nearest pole $p_{beta} \in P_{\mathcal{H}}$ s.t. $\vec{p_\alpha} \cdot \vec{p_\beta} = -1$ and for which the connection rules hold. An example of an initial configuration is shown in Figure 8.3.

There are no guarantees, and it is not necessary, that for each initial configuration the maximal match set will have cardinality equal to $|P_{\mathcal{M}_i}|$.

In order to explore all the possible match sets it is necessary to produce, for each maximal match set, the *best* subset for each cardinality.

What is sought is a similarity measure that is both position and orientation independent in space, able to quantify the similarity between both position and orientation of the set of the chosen poles of $\mathcal{M}_i$ and the set of the matched poles of $\mathcal{H}$. The proposed similarity measure is based on the computation of the euclidean and angular distances between poles.

Given a set $\mathcal{S}_i^k$ of matches (p_a, p_b), two undirected graphs $\mathcal{G}_\mathcal{M} = (V_\mathcal{M}, E_\mathcal{M})$ and $\mathcal{G}_\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$, where to each node $v_a^\mathcal{M} \in V_\mathcal{M}$ corresponds a pole $p_a \in P_{\mathcal{M}_i}$ and to each node $v_b^\mathcal{H} \in V_\mathcal{H}$ corresponds a pole $p_b \in P_\mathcal{H}$,

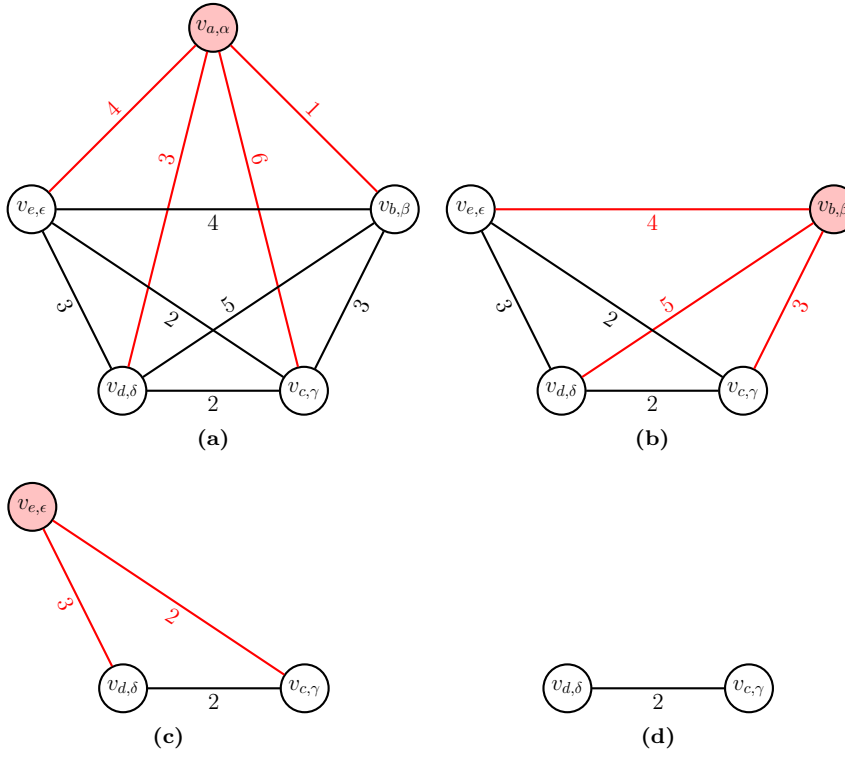


Figure 8.4: *Graph operations to obtain, from a maximal match set, the* best *subsets. To each arc is associated a cost (here with dummy values). In red are highlighted the nodes with highest cost and their outgoing arcs, that will be removed at the next step.*

Both graphs are complete and the cost associated to each arc (v_r, v_s) is calculated as:

$$\mathcal{C}_{r,s} = D_e(\dot{p_r}, \dot{p_s}) + ((\vec{p_r} \cdot \vec{p_s}) + 1).$$

where: $D_e(\dot{p_r}, \dot{p_s})$ is the Euclidean distance normalized by the bounding box diagonal of $\mathcal{M}_i$ and $(\vec{p_r} \cdot \vec{p_s}) + 1$ guarantees that $\mathcal{C}_{i,j}$ has zero as its minimum value. Depending on the graph to which is referred to, the cost will be denoted as $\mathcal{C}^{\mathcal{H}}_{r,s}$ or $\mathcal{C}^{\mathcal{M}}_{r,s}$. All the costs can be precomputed, since for each step of the composition the sets $P_{\mathcal{M}_i}$ and $P_{\mathcal{H}}$ remain unchanged until the connection happens.

After the two graphs have been built, a new graph $\mathcal{G}_d = (V_d, E_d)$ is built, where each of its nodes represents a match (p_a, p_b) and the cost of each arc is calculated as :

$$\mathcal{C}^d_{r,s} = |\mathcal{C}^H_{r,s} - \mathcal{C}^M_{r,s}|$$

The described cost quantifies how, the geometrical relationship between the poles p_r and p_s of $\mathcal{M}_i$ is similar to the one between their matched poles. Then for each node $v_r \in V_d$ the sum of the costs of its incident arcs is calculated as:

$$\mathcal{C}^d_{r,*} = \sum_s \mathcal{C}^d_{r,s}$$

the *best* subsets of the initial maximal set are then obtained iteratively removing from $\mathcal{G}_d$ the node v_r with larger $\mathcal{C}^d_{r,*}$, except for the subset of cardinality 1 for which the initial match is used. Figure 8.4 shows an example of this procedure. The generated subsets are then used as candidate match sets. At each step the values $\mathcal{C}^d_{r,*}$ are updated.

Finding the *optimal* match set

Once the list of all the candidate match sets is built, for each match set $\mathcal{S}^k_i$ a rigid transformation T^k that aligns $\mathcal{M}_i$ to $\mathcal{H}$ w.r.t. $\mathcal{S}^k_i$ is computed as defined in 8.3.1.

Since the aim is to find a match set that optimizes a given metric w.r.t the number of poles involved, a descriptor is computed for each match. This descriptor is defined upon euclidean distance, angular distance, and angular distance of the elongation direction between the matched poles. In this case the descriptor strictly depends on the actual position and orientation in space of the transformed module.

Given a match (p_a, p_b) the *cost* of using it to connect $\mathcal{M}_i$ to $\mathcal{H}$ is calculated as

$$\mathcal{D}_m(p_a, p_b) = D_e(\dot{p_a}, \dot{p_b}) + D_a(p_a, p_b)$$

where D_e is defined as the euclidean distance (normalized by the bounding box diagonal of $\mathcal{M}_i$) and:

$$D_a(p_a, p_b) = 2 - e^{-\frac{1}{2}\left(\frac{\vec{p_a}\cdot\vec{p_b}+1}{\sigma_1}\right)^2} - e^{-\frac{1}{2}\left(\frac{g(\overrightarrow{p_a},\overrightarrow{p_b})-1}{\sigma_2}\right)^2}$$

which is an adaptation of the so called *Fidelity Term* used in [LVS*13] and with g defined as :

$$g(\overrightarrow{p_a}, \overrightarrow{p_b}) = \begin{cases} 1 & \text{if cross-section is round} \\ \overrightarrow{p_a} \cdot \overrightarrow{p_b} & \text{if cross-section is elongated on one side} \\ |\overrightarrow{p_a} \cdot \overrightarrow{p_b}| & \text{if cross-section is elongated on both sides} \end{cases}$$

The parameters σ_1 and σ_2 are used to regulate the contribution of the two terms. During the performed experiments, it was usually preferred to strictly align the elongation direction $\overrightarrow{p_a}$, while allowing for small misalignments between the pole directions $\vec{p}$ using $\sigma_1 = 0.2$ and $\sigma_2 = 0.12$. The values of D_a is zero when both directions are perfectly aligned, having $\vec{p_a}$ and $\vec{p_b}$ opposite. The cost for each match set $\mathcal{S}_i^k$ is then calculated as

$$\mathcal{D}_m(S_k^i) = \frac{\sum_{a,b}(D_m(p_a, p_b))}{e^{c*n}}$$

where n is the cardinality of S_k^i and $c \in \mathbb{R}^+$ is a parameter used to define how much should be favored an increase of the number of matches used at the cost of a worse alignment. Usually $c = 1.5$ provided a good tradeoff in the experiments that have been carried out.

The match set with the lowest $\mathcal{D}_m(S_k^i)$ is then elected as the *optimal* match set. An instance of $\mathcal{M}_i$ is then generated, transformed using T_* and connected to $\mathcal{H}$ by means of the matches in S_k^*.

8.6 Collision detection

The PAM representation naturally encodes the shape's skeleton, which can be reified by collapsing all the rib edge loops and trivially connecting the nodes. The skeletal nodes of a PAM will be of type **Ln** in case of poles, type **Bn** in case of complex rib edge loops, that are the loops where different annular regions meet, and type **Jn** in case of simple rib edge loops, the one belonging to a single annular region. Nodes of type **Bn** and **Jn** are obtained collapsing the edge loop to its centroid. Two nodes are connected if their generating edge loops are connected by a single spine edge.

The one above is the description of the procedure used to obtain the representation of the skeleton of the modules of the toolbox and the one of the starter module. During the composition process, the skeleton of the module's instance being connected to the main structure, and the skeleton of the main structure itself are merged connecting together the poles of the

chosen match set; the **Ln**'s corresponding to the poles will be merged and converted into **Jn**'s and their radii updated. The described operation is similar to the copy and paste operation of the presented interactive tool described in Section 3.1.

Having this skeletal structure a simple skeleton based collision detection has been implemented, based on a bounding spheres hierarchy, from the rest of the section called BSH. The root of this hierarchy is the bounding sphere of the shape and the leaves are the maximal spheres centered at the skeletal nodes.

In order to obtain a more precise space partitioning, each branch of the skeleton needs to be resampled; starting from a **Bn** the branch is traversed, and for each pair of nodes (n_i, n_j the number of additional nodes needed is computed as $a_{i,j} = (D_e(n_i, n_j) - r_i - r_j)/(r_i + r_j)$; each value $a_{i,j}$ is rounded to the lowest integer or increased to zero if smaller. The sum of all the values $a_{i,j}$ gives the number of nodes that must be added to the considered branch to better fit space. This procedure is greedily run at each connection of a module. The actual resampling is done using the approach described in Section 3.2 preserving the original nodes of the branch.

The BSH is built top to bottom. The root of this hierarchy is the skeleton's bounding sphere, with a children centered each **Bn** and as radius the distance between the **Bn** and the farthest node belonging to its outgoing branches. Each **Bn** node of the hierarchy has a child for each outgoing branch; each branch can be the child of only one **Bn**, hence to lower the complexity, **Bn**'s are ordered from left to right in the hierarchy with increasing cardinality. Moreover each BSH node representing a **Bn**-**Bn** branch (B_i, B_j) is assigned to either the node of B_i or B_j in order to minimize the difference in number of children between them.

The radius associated to each branch node is equal to half of the maximal euclidean distance between any two nodes of that branch and the sphere is centered at the centroid of the nodes.

Branch nodes become then the root of a binary subtree composed recursively splitting in half each branch until each node of the BSH represents a single skeletal node.

Testing for collisions

Both the main structure $\mathcal{H}$ and each module $\mathcal{M}_i$ are provided with their BSH. The basic intersection test is to check if the distance between the centers of the spheres is smaller than the sum of their radii.

The collision test is performed traversing the BSH's starting from the roots. If the spheres at root nodes intersect, both hierarchies are traversed recursively testing only the children of nodes whose spheres intersect. If an intersection at the leaves is found the collision is reported.

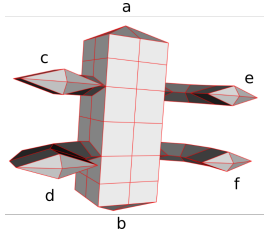
This BSH construction is similar to sphere-trees [Hub96] except that it

works solely on the skeleton instead of the geometry of the object, hence it has a lower number of primitives from which the hierarchy is built. However, for complex models, the traversal of the hierarchy could become inefficient due to the high number of paths on the hierarchy that must be traversed in order to test for a collision.

Chapter 9

Discussion

The presented method is still a work in progress, however it showed encouraging results. An experienced modeler partecipated to the experiments providing some feedbacks about the definition of the connection rules; they, while being sometimes tricky, appear to be intuitive and expressive enough to obtain predictable results.



To show how changes to the connection rules can affect the results, different sets of results have been obtained starting from a toolbox containing a single module for which different connection rules have been defined constraining the composition in order to produce different classes of shapes. Using just the module on the left, changing its pole's valence and elongation direction and selectively disallowing poles to be connected to themselves it is possible to build different kinds of shapes. Different results can be seen on Figures 9.1, 9.2, 9.4, and 9.5. Each result has undergone a subdivision step.

The procedure allows both to produce complex tangled, but intersection-free, space filling structures, using round cross sections and the same valence on every pole in order to allow a variety of configurations.

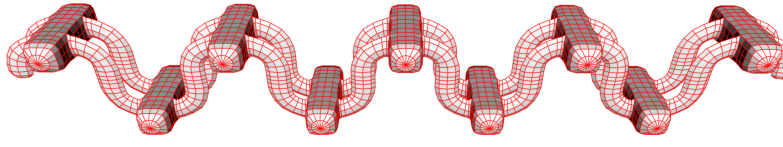


Figure 9.1: *In this example the poles* a *and* b *are not allowed to connect to themselves or to other poles.*

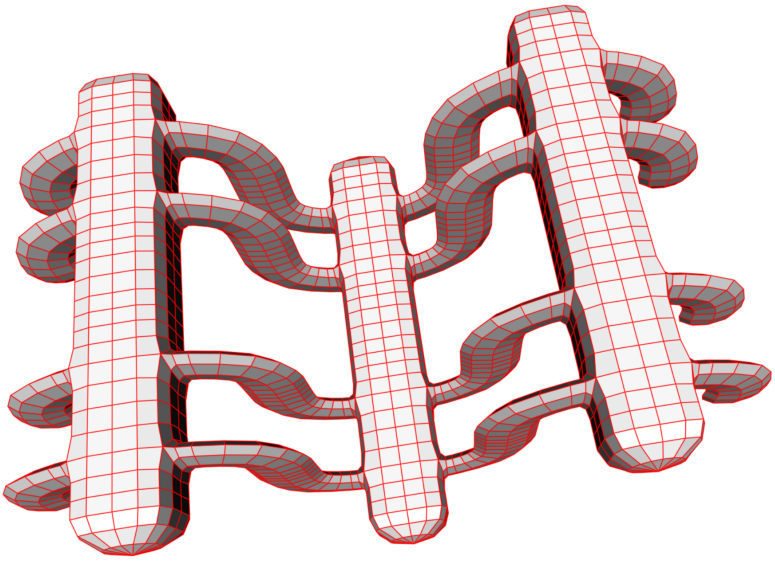


Figure 9.2: *In this example the poles* a *and* b *are allowed to connect with each other but not to themselves.*

Defining carefully crafted modules, the proposed method allows to produce highly regular shapes, with predictable results. An example is presented in Figure 9.3 where a structure similar to the Buckyball can be composed starting from a single module that represents a single atom of the original molecule.

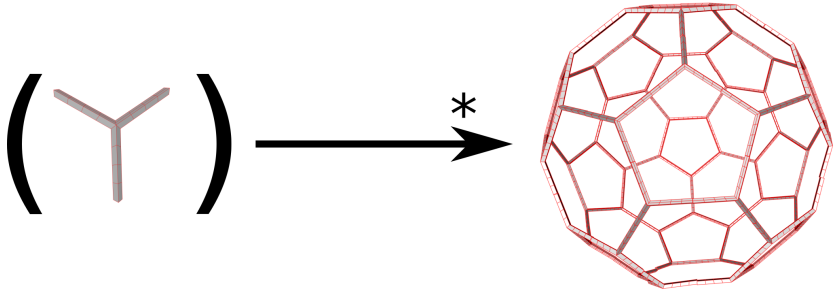
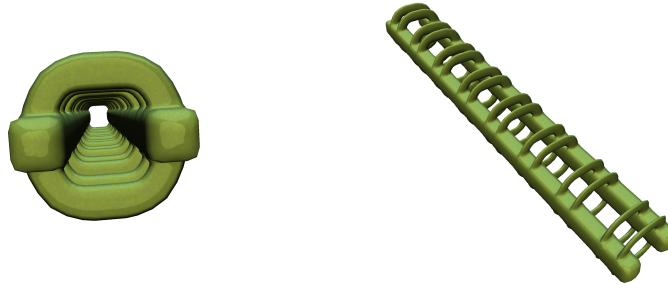
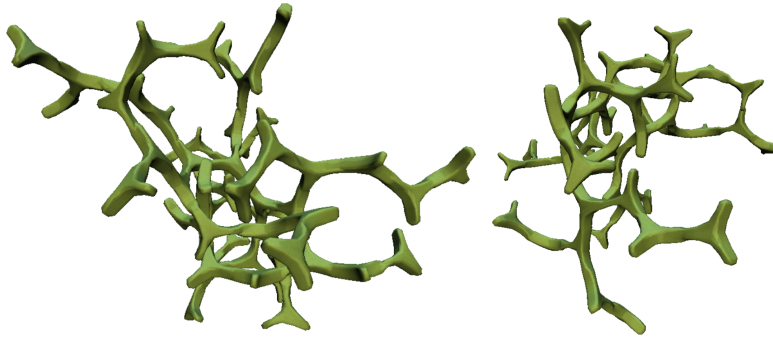


Figure 9.3: *The simple module on the left has been used to compose the regular shape on the right.*



(a) **(b)**

Figure 9.4: *Each polar cap have been shaped with the elongation on one side, in order to force the connection with a specific orientation.*



(a) *space filling structure* **(b)** *other view*

Figure 9.5: *Example of tangled structure with no self intersections*

Conclusions and Future Works

Chapter 10

Conclusions

The representation of digital objects based on triangular meshes is the *de facto* standard for interactive rendering, however, their simplicity and lack of a global structure becomes also a limit for some purposes, such as modeling or animation. Throughout this thesis, two different ways for obtaining structured meshes have been shown. The underpinning of both methods is the use of a shape descriptor, the curve-skeleton, able to provide geometrical, structural, and topological information. As shown in Part III, relying on purely geometric features (e.g. principal curvature directions) could be not enough to recover the object's global structure, while the use of the skeleton allows to produce quad layouts that faithfully partition the input shape into semantically meaningful patches that are also well aligned to the shape's features.

Another applicative context can benefit from the use of structured models is part based modeling, where the aim is to quickly build a shape, composing together different available models of *parts*, instead of modeling it from a scratch. These approaches usually rely on, and produce, triangular meshes, while the approach presented in Part IV uses and produces only structured models, represented as Polar Annular Meshes, using the information provided by the skeleton, naturally encoded in the PAM structure, to guide the composition process.

Finally have been observed throughout the development of the quad layout computation method, that the problem of generating high-quality curve-skeletons is still open. While state-of-the-art methods are capable of extracting meaningful skeletons from complex shapes they often fail at understanding the shape at a higher level, generating a robust one-to-one correspondence with its logical components. Hence the resulting skeletons are not always directly usable in any pipeline without further processing. While some of the issues present in automatically extracted skeletons can be solved algorithmically, others, more tied to the shape semantics are more

challenging, but can be efficiently solved with the user intervention, with the use of an interactive tool, like the one proposed in Part II.

Interactive Curve-Skeleton Processing

The evaluation of what is a good curve-skeleton is strictly dependent to the application in which it will be used. Robust state-of-the-art skeletonization methods exist, but they often, produce results that are affected by a number of issues, mainly due to noise; moreover, a fine-tuning of their input parameters is often necessary to obtain better results, but is not always resolutive in the most challenging cases. These issues prevent them to be used without processing in any pipeline guaranteeing optimal results.

In response to these observations and practical issues, SkeletonLab has been developed. It has been observed that, in order to obtain optimal results from pipelines that utilize curve-skeleton, an effective and practical approach can be to automatically extract the curve-skeleton with the existing methods, using a standard parameter setting and process it manually in order to optimally fit the pipeline. Moreover the presented tool can be useful in an explorative phase where the requirements and desired properties of the skeleton are not known. SkeletonLab has been successfully used to repair some of the input skeletons used in the quad layout extraction method presented in Part III.

Quad Layout Extraction

The quad layout computation method presented in this thesis uses the curve-skeleton of a triangle mesh as a guidance to compute its coarse quad layout; the shape segmentation induced by the skeleton is used to define both the location of the irregular vertices and their connectivity. The method is simple to use, can run in a fully automatic way, but supports friendly user interaction using an intermediate structure, called *Fattened Skeleton*, which overcomes the difficulties of directly editing a quad layout, which is known to be a complex task even for experienced users, due to the global effects of local operations. The presented method is also fast, in fact, interactive sessions can be completed in few minutes, supporting user refinements at interactive frame rates. The use of the curve-skeleton as guidance allows to capture the structure of the original shape and to cover its surface with quad domains that are aligned to its elongated parts; moreover, differently from other methods which usually first define the location of irregular vertices and then their connectivity, in this method the global structure of the quad layout is defined independently from the surface, according solely to the skeleton, and it is then mapped onto it.

The produced layouts support easy extraction of semi-regular pure quad meshes at user-defined resolution, and domains for UV-mapping. The layouts (and the quad meshes) generated with this method have few singularities of low valence and are usually coarser then the ones obtained with previous approaches.

Composition of PAMs

In the last part of this thesis, the generation of structured meshes has been addressed from a different point of view and with the use of a different mesh representation. Instead of converting an unstructured triangle mesh to a structured quad layout, the proposed approach starts from a set of building blocks (the *modules*) represented using Polar Annular Meshes(PAMs), which are structured quad-dominant meshes, and procedurally composes them together guaranteeing that the resulting mesh is still structured.

This approach makes extensive use of the skeleton of the modules used for the composition, both for deciding how to connect them to the shape being composed and to define bounding volume hierarchies for collision detection purposes.

While the proposed method is still a work in progress, it has shown encouraging results. It is suitable for modeling both organic and regular shapes and the proposed approach of encoding the connection rules in the geometry at the polar regions appears to be rather expressive and friendly to use for modelers and can be easily integrated in their workflow. This compositional approach can be used both for producing inspiring shapes and base models that can be subsequently refined. The resulting shapes are always valid PAMs thus encoding their overall structure and their skeleton, allowing them to be easily segmented and animated.

Similarities between PAMs and skeletally guided Quad Layouts

Starting from the perspective of having a skeletally supported structured mesh representation, it can be seen that both quad layouts and PAMs obtained with the presented methods are similar. In both cases their geometry and structure are strictly related to the topological and geometrical information conveyed by the skeleton, suggesting that, for example two quad meshes obtained from the quad layouts computed with the proposed method, can be connected together preserving their *structuredness*, similarly to the way in which two PAMs can be connected together using their poles. The operation would require to cut the two quad meshes open at a loop of four valence 3 vertices and merge the two loops together, then refining the para-

meterization using the approach seen in Sections 6.2.7. Both operations are equivalent to merge two **Ln**'s nodes of the underlying skeleton of the two objects. Moreover, the PAM's primitive operation of adding/removing a PAM feature presented in Section can be transposed to the presented quad layout structure using simple geometrical operations as depicted in Figure 10.1, which in turn is equivalent to create a new branch of the skeleton.

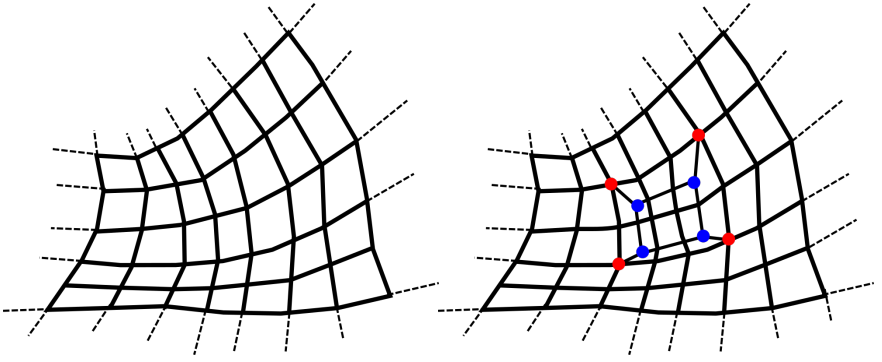


Figure 10.1: *A quad layout build around a curve-skeleton (a), as proposed in Part III of this thesis can be trivially modified, adding a skeletally supported feature (b); similarly to what can be done with PAMs.*

This leads to the conclusion that both representations exhibit the same relation to the skeleton of the shape and can be used for similar purposes; for example a procedural composition approach, similar to the one proposed for PAMs, can be defined using semi-regular quad meshes obtained from the presented quad layout structure.

Future Works

Quad Layout Extraction

There are several interesting extensions that can be added to this interactive system. Following [BAS14], it can be extended to a modeling system, by giving the user the possibility to edit the curve-skeleton associated to the original mesh, extruding branches, cloning parts, or specifying the properties of each branch (e.g., the radii of medial balls). The interactive session can use symmetry planes or points to modify the skeleton and, thus, the base complex and the quad layout. Such modeling system could also support part-based modeling, that could also be performed in a procedural fashion.

A slightly modified version of this method that would take as input only a skeleton with associated sphere radii can be used as an inverse skeletonization method guaranteeing to produce all quad base meshes with a proper structure.

The base complex can be used to model a cage around the shape for animation purposes. Or it can be taken as the control mesh of a subdivision surface: optimal positions of nodes of the control grid can be found easily, such that the limit surface fits the original mesh; fine detail can be easily added via displacement mapping. Consistent quad layouts of different poses of the same object can be easily obtained: this provides the basis for a cross-parametrization, which can be used for the purpose of animation. The quad layout $\mathcal{Q}$ resulting from this method can be also seen as the outer surface of a solid mesh of hexahedra: in order to obtain a geometric hexahedral mesh, one should assign 3D positions to all the internal vertices (e.g., by computing harmonic functions that use the coordinates of the boundary vertices as Dirichlet boundary conditions).

Finally, would be of interest to further investigate the problem of curve-skeleton extraction, in order to benefit the presented algorithm as well as any other method that requires such a high level of abstraction of a 3D shape.

Procedural PAM Composition

The proposed method, being a work in progress, is open to a number of refinements, modifications and evolutions. A editor for *modules* can be introduced to support the user, once the module has been created, to an easier definition of the connection rules; a set of simple operations for creating and editing the toolboxes can be a useful addition as well.

The definition of the rules can be extended with the pole's colorization to encode additional information. The composition process as defined favours rigid connections; in order to make it more suitable to generate organic shapes, it can be modified in order to tolerate a wider misalignment between pole's directions and, after the connection has been established, smoothing the portions of the skeleton that have been involved, modifying the surrounding geometry accordingly. The composition process can also be modified to take into account the *age* of the poles of the main structure, favouring the older ones, in order to obtain a more harmonic and balanced composition of the shape.

Bibliography

[ABZ*] Alliez P., Bala K., Zhou K., Liu L., Chambers E. W., Letscher D., Ju T.: A simple and robust thinning algorithm on cell complexes.

[ATC*08] Au O. K.-C., Tai C.-L., Chu H.-K., Cohen-Or D., Lee T.-Y.: Skeleton Extraction by Mesh Contraction. *ACM Trans. Graph. 27*, 3 (2008), 44:1–44:10.

[Aut07] Autodesk: Mudbox, 2007. `http://www.autodesk.com`.

[Aut15] AutoDesk 123D:. `http://www.123dapp.com/`, 2015.

[BAS14] Bærentzen J. A., Abdrashitov R., Singh K.: Interactive Shape Modeling Using a Skeleton-mesh Co-representation. *ACM Trans. Graph. 33*, 4 (July 2014), 132:1–132:10.

[BCE*13] Bommes D., Campen M., Ebke H.-C., Alliez P., Kobbelt L.: Integer-grid maps for reliable quad meshing. *ACM Trans. Graph. 32*, 4 (July 2013), 98:1–98:12.

[BJD*12] Borosan P., Jin M., DeCarlo D., Gingold Y., Nealen A.: RigMesh: Automatic rigging for part-based shape modeling and deformation. *ACM Transactions on Graphics (TOG) 31*, 6 (Nov. 2012), 198:1–198:9.

[BLK11] Bommes D., Lempfer T., Kobbelt L.: Global Structure Optimization of Quadrilateral Meshes. *Comput. Graph. Forum 30*, 2 (2011), 375–384.

[BLP*13] Bommes D., Lévy B., Pietroni N., Puppo E., Silva C., Tarini M., Zorin D.: Quad-Mesh Generation and Processing: A Survey. *Comput. Graph. Forum 32* (2013), 51–76.

[Blu67] Blum H.: A Transformation for Extracting New Descriptors of Shape. *Models for the Perception of Speech and Visual Form* (1967), 362–380.

[BMUS15] Barbieri S., Meloni P., Usai F., Scateni R.: Skeleton Lab: an Interactive Tool to Create, Edit, and Repair Curve-Skeletons. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference* (2015), Giachetti A., Biasotti S., Tarini M., (Eds.), The Eurographics Association.

[BMW12] Bærentzen J., Misztal M., Wełnicka K.: Converting skeletal structures to quad dominant meshes. *Computers & Graphics 36*, 5 (2012), 555 – 561. Shape Modeling International (SMI) Conference 2012.

[Bom12] Bommes D.: *Quadrilateral Surafce Mesh Generation for Animation and Simulation*. Shaker Verlag, 2012.

[Bor96] Borgefors G.: On digital distance transforms in three dimensions. *Computer Vision and Image Understanding 64*, 3 (1996), 368 – 376.

[BP07] Baran I., Popović J.: Automatic rigging and animation of 3d characters. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM.

[BVK10] Bommes D., Vossemer T., Kobbelt L.: Quadrangular Parameterization for Reverse Engineering. In *Proceedings of the 7th International Conference on Mathematical Methods for Curves and Surfaces* (2010), MMCS'08, Springer-Verlag, pp. 55–69.

[BWR*08] Bergou M., Wardetzky M., Robinson S., Audoly B., Grinspun E.: Discrete Elastic Rods. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 63:1–63:12.

[BZK09] Bommes D., Zimmer H., Kobbelt L.: Mixed-integer Quadrangulation. *ACM Trans. Graph. 28*, 3 (2009), 77:1–77:10.

[CBK12] Campen M., Bommes D., Kobbelt L.: Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph. 31*, 4 (2012), 110:1–110:11.

[CC78] Catmull E., Clark J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350 – 355.

[CGA15] CGAL, Computational Geometry Algorithms Library:. `http://www.cgal.org`, 2015.

[CK14a] Campen M., Kobbelt L.: Dual Strip Weaving: Interactive Design of Quad Layouts Using Elastica Strips. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 183:1–183:10.

[CK14b] Campen M., Kobbelt L.: Quad layout embedding via aligned parameterization. *Computer Graphics Forum* (2014), n/a–n/a.

[CKGK11] Chaudhuri S., Kalogerakis E., Guibas L., Koltun V.: Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph. 30*, 4 (July 2011), 35:1–35:10.

[CSM07] Cornea N. D., Silver D., Min P.: Curve-Skeleton Properties, Applications, and Algorithms. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (May 2007), 530–548.

[CSYB05] Cornea N., Silver D., Yuan X., Balasubramanian R.: Computing hierarchical curve-skeletons of 3D objects. *The Visual Computer 21*, 11 (October 2005), 945–955.

[CTO*10] Cao J., Tagliasacchi A., Olson M., Zhang H., Su Z.: Point cloud skeletons via Laplacian based contraction. In *SMI 2010* (June 2010), pp. 187–197.

[DHL*98] Deussen O., Hanrahan P., Lintermann B., Měch R., Pharr M., Prusinkiewicz P.: Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 275–286.

[DS06] Dey T. K., Sun J.: Defining and Computing Curve-skeletons with Medial Geodesic Function. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (2006), SGP '06, pp. 143–152.

[EAVD79] El-Attar R., Vidyasagar M., Dutta S.: An Algorithm for l_1-Norm Minimization with Application to Nonlinear l_1-Approximation. *SIAM Journal on Numerical Analysis 16*, 1 (1979), 70–86.

[EBCK13] Ebke H.-C., Bommes D., Campen M., Kobbelt L.: QEx: Robust Quad Mesh Extraction. *ACM Trans. Graph. 32*, 6 (2013), 168:1–168:10.

[ECBK14] Ebke H.-C., Campen M., Bommes D., Kobbelt L.: Level-of-detail quad meshing. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 184:1–184:11.

[EGKT08] Eppstein D., Goodrich M. T., Kim E., Tamstorf R.: Motorcycle graphs: Canonical quad mesh partitioning. *Computer Graphics Forum 27*, 5 (2008), 1477–1486.

[FKS*04] Funkhouser T., Kazhdan M., Shilane P., Min P., Kiefer W., Tal A., Rusinkiewicz S., Dobkin D.: Modeling by example. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 652–663.

[Flo03] Floater M. S.: Mean value coordinates. *Computer aided geometric design 20*, 1 (2003), 19–27.

[Gam08] Games E.: Spore, 2008.

[GK04] Giblin P., Kimia B. B.: A formal classification of 3d medial axis points and their local geometry. *IEEE Trans Pattern Anal Mach Intell 26*, 2 (Feb 2004), 238–251.

[GLXJ14] Guo X., Lin J., Xu K., Jin X.: Creature grammar for creative modeling of 3d monsters. *Graphical Models 76* (2014), 376–389.

[GSZ11] Gregson J., Sheffer A., Zhang E.: All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2011) 30*, 5 (2011).

[Gur13] Gurobi Optimization: `http://www.gurobi.com`, 2013.

[Hav05] Havemann S.: *Generative Mesh Modeling / Havemann, Sven.* PhD thesis, 2005.

[Hec94] Heckbert P. S.: Graphics Gems IV. Academic Press Professional, Inc., San Diego, CA, USA, 1994, ch. Bilinear Coons Patch Image Warping, pp. 438–446.

[HJS*14] Huang J., Jiang T., Shi Z., Tong Y., Bao H., Desbrun M.: ℓ1-based Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph. 33*, 3 (June 2014), 25:1–25:11.

[HSKK01] Hilaga M., Shinagawa Y., Kohmura T., Kunii T. L.: Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 203–212.

[Hub96] Hubbard P. M.: Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph. 15*, 3 (July 1996), 179–210.

[JLW10] Ji Z., Liu L., Wang Y.: B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes. *Computer Graphics Forum (Proceedings of Pacific Graphics) 29*, 7 (2010), 2169–2178.

[KJT13] Kustra J., Jalba A., Telea A.: Probabilistic View-based 3D Curve Skeleton Computation on the GPU. 237–246.

[KLS03] Khodakovsky A., Litke N., Schröder P.: Globally Smooth Parameterizations with Low Distortion. *ACM Trans. Graph. 22*, 3 (July 2003), 350–357.

[KNP07] Kaelberer F., Nieser M., Polthier K.: QuadCover - Surface Parameterization using Branched Coverings. *Computer Graphics Forum* (2007).

[LGS12] Livesu M., Guggeri F., Scateni R.: Reconstructing the Curve-Skeletons of 3D Shapes Using the Visual Hull. *IEEE Transactions on Visualization and Computer Graphics 18*, 11 (2012), 1891–1901.

[LPRM02] Lévy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3 (July 2002), 362–371.

[LS13] Livesu M., Scateni R.: Extracting Curve-skeletons from Digital Shapes Using Occluding Contours. *The Visual Computer 29*, 9 (2013), 907–916.

[LVS*13] Livesu M., Vining N., Sheffer A., Gregson J., Scateni R.: PolyCut: Monotone Graph-Cuts for PolyCube Base-Complex Construction. *ACM Trans. Graph. 32*, 6 (2013), 171:1–171:12.

[LZLW15] Liu L., Zhang Y., Liu Y., Wang W.: Feature-preserving t-mesh construction using skeleton-based polycubes. *Computer-Aided Design* (2015), 162–172.

[Mor81] Morgenthaler D.: Three-dimensional simple points: serial erosion, parallel thinning and skeletonization. *Computer Vision Lab, Univ. of Maryland* (1981).

[MP96] Měch R., Prusinkiewicz P.: Visual models of plants interacting with their environment. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 397–410.

[MPZ14] Myles A., Pietroni N., Zorin D.: Robust field-aligned global parametrization. *ACM Trans. Graph. 33*, 4 (July 2014), 135:1–135:14.

[MS96] Ma C., Sonka M.: A fully parallel 3d thinning algorithm and its applications. *Computer Vision and Image Understanding 64*, 3 (1996), 420 – 433.

[MTP*15] Marcias G., Takayama K., Pietroni N., Panozzo D., Sorkine-Hornung O., Puppo E., Cignoni P.: Data-driven interactive quadrangulation. *ACM Trans. Graph. 34*, 4 (July 2015), 65:1–65:10.

[MWH*06] Müller P., Wonka P., Haegler S., Ulmer A., Van Gool L.: Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (July 2006), 614–623.

[PBJW14] Peng C.-H., Barton M., Jiang C., Wonka P.: Exploring Quadrangulations. *ACM Trans. Graph. 33*, 1 (2014), 12:1–12:13.

[Pil09] Pilgway: 3D-Coat, 2009. `http://3d-coat.com`.

[Pix07] Pixologic: ZBrush, 2007. `http://pixologic.com`.

[PL90] Prusinkiewicz P., Lindenmayer A.: *The algorithmic beauty of plants*. New York, NY, USA, 1990.

[PM01] Parish Y. I. H., Müller P.: Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 301–308.

[PPTSH14] Panozzo D., Puppo E., Tarini M., Sorkine-Hornung O.: Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Trans. on Graphics 33*, 4 (2014), 134.

[PTC10] Pietroni N., Tarini M., Cignoni P.: Almost isometric mesh parameterization through abstract domains. *IEEE Transactions on Visualization and Computer Graphics 16*, 4 (2010), 621–635.

[RLL*06] Ray N., Li W. C., Lévy B., Sheffer A., Alliez P.: Periodic global parameterization. *ACM Trans. Graph. 25*, 4 (Oct. 2006), 1460–1485.

[RRP15] Razafindrazaka F. H., Reitebuch U., Polthier K.: Perfect matching quad layouts for manifold meshes. *Computer Graphics Forum (proceedings of EUROGRAPHICS Symposium on Geometry Processing) 34*, 5 (2015).

[SGSG71] Stiny G., Gips J., Stiny G., Gips J.: Shape grammars and the generative specification of painting and sculpture. In *Segmentation of Buildings for 3DGeneralisation. In: Proceedings of the Workshop on generalisation and multiple representation , Leicester* (1971).

[SH09] Sorkine-Hornung O.: *Least-Squares Rigid Motion Using SVD*. Tech. rep., 2009.

[SMA] Srinivasan V., M E., Akleman E.: Solidifying wireframes.

[SS10] Schmidt R., Singh K.: Meshmixer: An interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 6:1–6:1.

[TACSD06] Tong Y., Alliez P., Cohen-Steiner D., Desbrun M.: Designing Quadrangulations with Discrete Harmonic Forms. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (2006), SGP '06, Eurographics Association, pp. 201–210.

[TAOZ12] Tagliasacchi A., Alhashim I., Olson M., Zhang H.: Mean Curvature Skeletons. *Comp. Graph. Forum 31*, 5 (2012), 1735–1744.

[THCM04] Tarini M., Hormann K., Cignoni P., Montani C.: PolyCube-Maps. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 853–860.

[TLL*11] Talton J. O., Lou Y., Lesser S., Duke J., Měch R., Koltun V.: Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2 (Apr. 2011).

[TPC*10] Tarini M., Pietroni N., Cignoni P., Panozzo D., Puppo E.: Practical quad mesh simplification. *Computer Graphics Forum (Special Issue of Eurographics 2010 Conference) 29*, 2 (2010), 407–418.

[TPHS13] Takayama K., Panozzo D., Hornung A., Sorkine O.: Sketch-based Generation and Editing of Quad Meshes. *ACM Trans. Graph. 32*, 4 (2013), 97:1–97:8.

[TPP*11] Tarini M., Puppo E., Panozzo D., Pietroni N., Cignoni P.: Simple quad domains for field aligned mesh parametrization. *ACM Trans. Graph. 30*, 6 (2011), 142:1–142:12.

[TPSH14] Takayama K., Panozzo D., Sorkine-Hornung O.: Pattern-Based Quadrangulation for N-sided patches. *Comp. Graph. Forum (Proceedings of SGP) 33*, 5 (2014), 177–184.

[TZCO09] Tagliasacchi A., Zhang H., Cohen-Or D.: Curve Skeleton Extraction from Incomplete Point Cloud. *ACM Trans. Graph. 28*, 3 (July 2009), 71:1–71:9.

[Usa15] Usai, Francesco and Livesu, Marco and Puppo, Enrico and Tarini, Marco and Scateni, Riccardo: Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. *ACM Trans. Graph. 35*, 1 (Dec. 2015), 6:1–6:13.

[WB06] Wächter A., Biegler L. T.: On the Implementation of an Interior-point Filter Line-search Algorithm for Large-scale Nonlinear Programming. *Math. Program. 106*, 1 (2006), 25–57.

[WB07] Wang T., Basu A.: A note on 'a fully parallel 3d thinning algorithm and its applications'. *Pattern Recognition Letters 28*, 4 (2007), 501 – 506.

[ZBG*07a] Zhang Y., Bazilevs Y., Goswami S., Bajaj C. L., Hughes T. J.: Patient-specific vascular {NURBS} modeling for isogeometric analysis of blood flow. *Computer Methods in Applied Mechanics and Engineering 196*, 29–30 (2007), 2943 – 2959.

[ZBG*07b] Zhang Y., Bazilevs Y., Goswami S., Bajaj C. L., Hughes T. J.: Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. *Computer methods in applied mechanics and engineering 196*, 29 (2007), 2943–2959.

[ZBr15] ZBrush ZSpheres:. `http://pixologic.com/zbrush/features/zspheres/`, 2015.

[ZCOM13] Zheng Y., Cohen-Or D., Mitra N. J.: Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum (Eurographics) 32*, 2pt2 (2013), 195–204.

[ZJY15] Zhu X., Jin X., You L.: Analytical solutions for tree-like structure modelling using subdivision surfaces. *Computer Animation and Virtual Worlds 26*, 1 (2015), 29–42.

[ZLLJ08] Zhang X., Liu J., Li Z., Jaeger M.: Volume decomposition and hierarchical skeletonization. In *Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry* (New York, NY, USA, 2008), VRCAI '08, ACM, pp. 17:1–17:6.